Load The Dataset (Week 2)

In [1]:
```
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
# ingest data
df=pd.read_csv('https://raw.githubusercontent.com/Christine971224/Analytics-2023/master/
df.head()
```

Out[1]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival
0	Resort Hotel	0	342	2015	July	27	
1	Resort Hotel	0	737	2015	July	27	
2	Resort Hotel	0	7	2015	July	27	
3	Resort Hotel	0	13	2015	July	27	
4	Resort Hotel	0	14	2015	July	27	

5 rows × 36 columns

In [2]:
```
#basic information of dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 119390 entries, 0 to 119389
Data columns (total 36 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   hotel                           119390 non-null  object
 1   is_canceled                     119390 non-null  int64
 2   lead_time                       119390 non-null  int64
 3   arrival_date_year               119390 non-null  int64
 4   arrival_date_month              119390 non-null  object
 5   arrival_date_week_number        119390 non-null  int64
 6   arrival_date_day_of_month       119390 non-null  int64
 7   stays_in_weekend_nights         119390 non-null  int64
 8   stays_in_week_nights            119390 non-null  int64
 9   adults                          119390 non-null  int64
 10  children                        119386 non-null  float64
 11  babies                          119390 non-null  int64
```

```
 12  meal                            119390 non-null  object 
 13  country                         118902 non-null  object 
 14  market_segment                  119390 non-null  object 
 15  distribution_channel            119390 non-null  object 
 16  is_repeated_guest               119390 non-null  int64  
 17  previous_cancellations          119390 non-null  int64  
 18  previous_bookings_not_canceled  119390 non-null  int64  
 19  reserved_room_type              119390 non-null  object 
 20  assigned_room_type              119390 non-null  object 
 21  booking_changes                 119390 non-null  int64  
 22  deposit_type                    119390 non-null  object 
 23  agent                           103050 non-null  float64
 24  company                         6797 non-null    float64
 25  days_in_waiting_list            119390 non-null  int64  
 26  customer_type                   119390 non-null  object 
 27  adr                             119390 non-null  float64
 28  required_car_parking_spaces     119390 non-null  int64  
 29  total_of_special_requests       119390 non-null  int64  
 30  reservation_status              119390 non-null  object 
 31  reservation_status_date         119390 non-null  object 
 32  name                            119390 non-null  object 
 33  email                           119390 non-null  object 
 34  phone-number                    119390 non-null  object 
 35  credit_card                     119390 non-null  object 
dtypes: float64(4), int64(16), object(16)
memory usage: 32.8+ MB
```

In [3]:

```
# View the proportion of empty values in each column
df.isnull().mean()
```

Out[3]:

```
hotel                             0.000000
is_canceled                       0.000000
lead_time                         0.000000
arrival_date_year                 0.000000
arrival_date_month                0.000000
arrival_date_week_number          0.000000
arrival_date_day_of_month         0.000000
stays_in_weekend_nights           0.000000
stays_in_week_nights              0.000000
adults                            0.000000
children                          0.000034
babies                            0.000000
meal                              0.000000
country                           0.004087
market_segment                    0.000000
distribution_channel              0.000000
is_repeated_guest                 0.000000
previous_cancellations            0.000000
previous_bookings_not_canceled    0.000000
reserved_room_type                0.000000
assigned_room_type                0.000000
booking_changes                   0.000000
deposit_type                      0.000000
agent                             0.136862
company                           0.943069
days_in_waiting_list              0.000000
customer_type                     0.000000
adr                               0.000000
required_car_parking_spaces       0.000000
```

```
total_of_special_requests        0.000000
reservation_status               0.000000
reservation_status_date          0.000000
name                             0.000000
email                            0.000000
phone-number                     0.000000
credit_card                      0.000000
dtype: float64
```

In [4]:

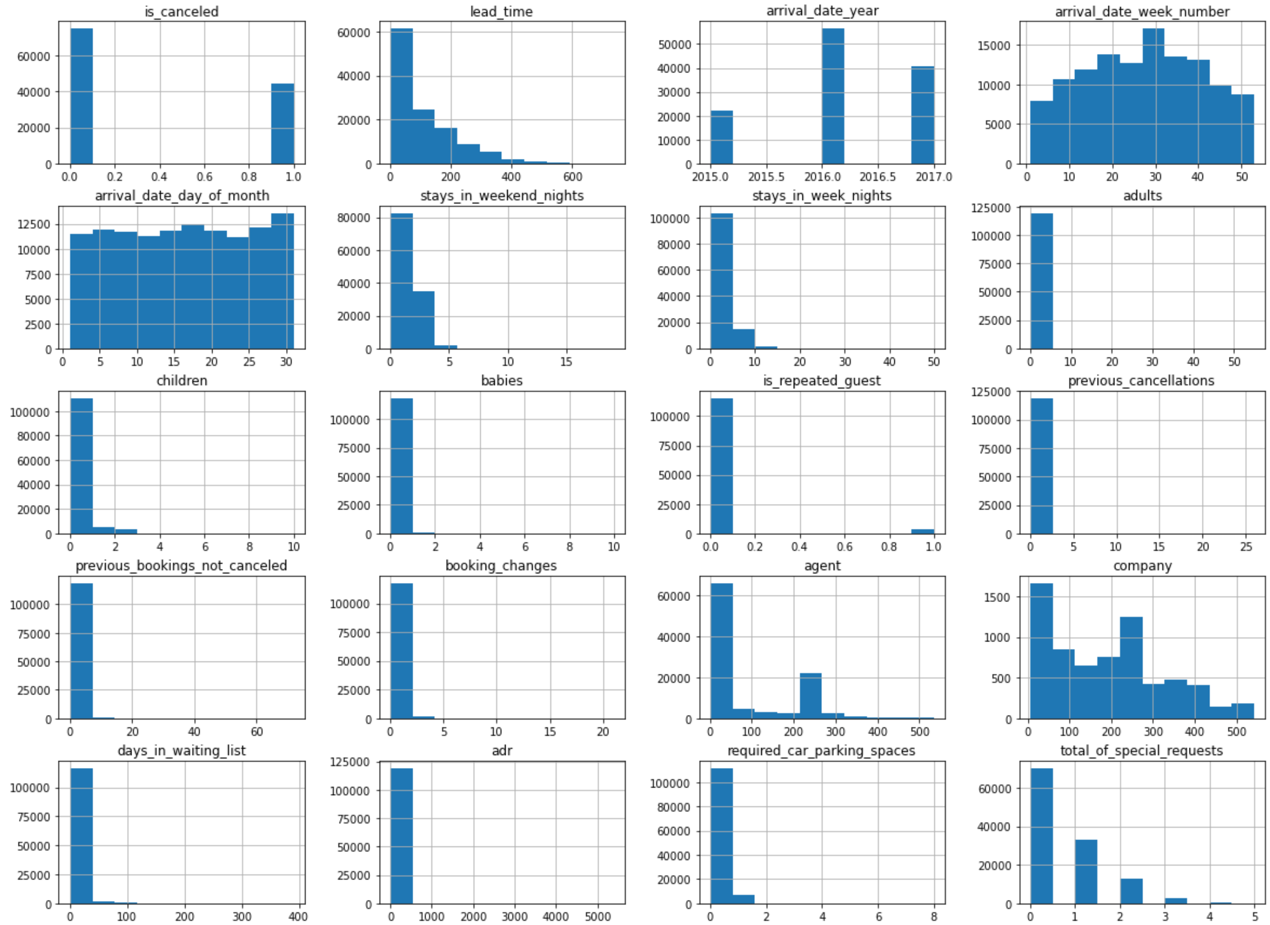
```
# Transpose the resulting DataFrame
df.describe([0.01,0.05,0.1,0.25,0.5,0.75,0.99]).T
```

Out[4]:

	count	mean	std	min	1%	5%	10%	2
is_canceled	119390.0	0.370416	0.482918	0.00	0.0	0.0	0.0	0.
lead_time	119390.0	104.011416	106.863097	0.00	0.0	0.0	3.0	18.
arrival_date_year	119390.0	2016.156554	0.707476	2015.00	2015.0	2015.0	2015.0	2016.
arrival_date_week_number	119390.0	27.165173	13.605138	1.00	2.0	5.0	8.0	16.
arrival_date_day_of_month	119390.0	15.798241	8.780829	1.00	1.0	2.0	4.0	8.
stays_in_weekend_nights	119390.0	0.927599	0.998613	0.00	0.0	0.0	0.0	0.
stays_in_week_nights	119390.0	2.500302	1.908286	0.00	0.0	0.0	1.0	1.
adults	119390.0	1.856403	0.579261	0.00	1.0	1.0	1.0	2.
children	119386.0	0.103890	0.398561	0.00	0.0	0.0	0.0	0.
babies	119390.0	0.007949	0.097436	0.00	0.0	0.0	0.0	0.
is_repeated_guest	119390.0	0.031912	0.175767	0.00	0.0	0.0	0.0	0.
previous_cancellations	119390.0	0.087118	0.844336	0.00	0.0	0.0	0.0	0.
previous_bookings_not_canceled	119390.0	0.137097	1.497437	0.00	0.0	0.0	0.0	0.
booking_changes	119390.0	0.221124	0.652306	0.00	0.0	0.0	0.0	0.
agent	103050.0	86.693382	110.774548	1.00	1.0	1.0	6.0	9.
company	6797.0	189.266735	131.655015	6.00	16.0	40.0	40.0	62.
days_in_waiting_list	119390.0	2.321149	17.594721	0.00	0.0	0.0	0.0	0.
adr	119390.0	101.831122	50.535790	-6.38	0.0	38.4	50.0	69.
required_car_parking_spaces	119390.0	0.062518	0.245291	0.00	0.0	0.0	0.0	0.
total_of_special_requests	119390.0	0.571363	0.792798	0.00	0.0	0.0	0.0	0.

In [5]:

```
import matplotlib.pyplot as plt
# generate histograms for all the columns
df.hist(figsize=(20,15))
plt.show()
```



EDA (Week 3)

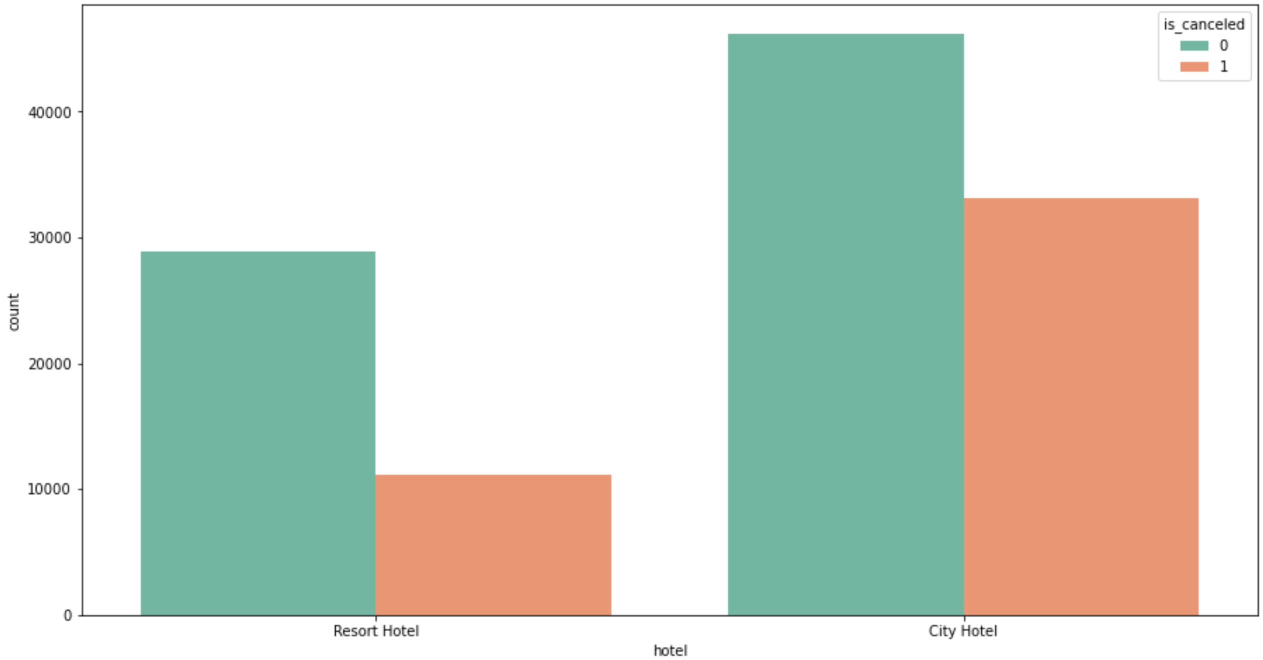
1. Hotel bookings and cancellations

In [6]:

```
# The number of hotel reservations and cancellations can directly show the actual numbe
import seaborn as sns
plt.figure(figsize=(15,8))
sns.countplot(x='hotel'
             ,data=df
             ,hue='is_canceled'
             ,palette=sns.color_palette('Set2',2)
            )
```

Out[6]:

```
<AxesSubplot:xlabel='hotel', ylabel='count'>
```



In [7]:
```
#calculate the proportion of cancellations for each unique value in the 'hotel' column (
hotel_cancel=(df.loc[df['is_canceled']==1]['hotel'].value_counts()/df['hotel'].value_co
print('Hotel cancellations'.center(20),hotel_cancel,sep='\n')
```

```
Hotel cancellations
City Hotel      0.417270
Resort Hotel    0.277634
Name: hotel, dtype: float64
```

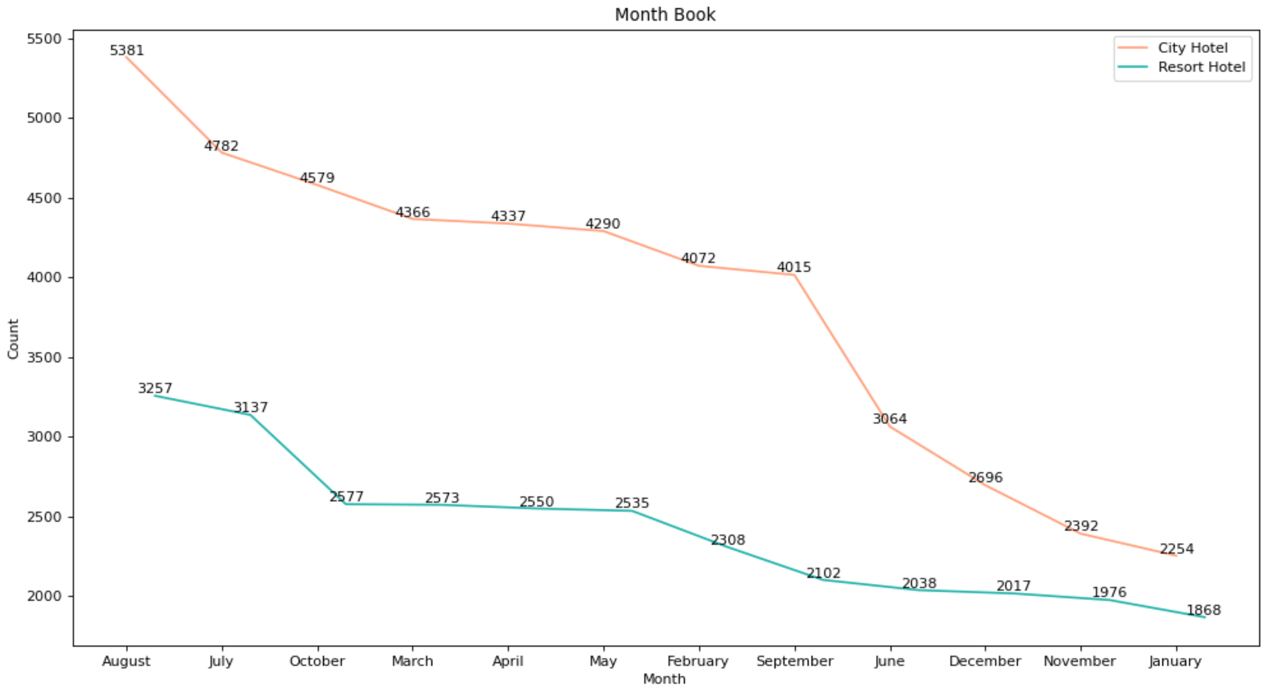
Comment: City Hotel's booking volume and cancellation volume are both higher than Resort Hotel's, but Resort Hotel's cancellation rate is 27.8%, while City Hotel's cancellation rate reaches 41.7%.

2. Hotel bookings by month

In [8]:
```
#create a plot to visualize the number of bookings for "City Hotel" and "Resort Hotel" (
#focusing only on the ones where 'is_canceled' is 0, meaning the bookings that were not
city_hotel=df[(df['hotel']=='City Hotel') & (df['is_canceled']==0)]
resort_hotel=df[(df['hotel']=='Resort Hotel') & (df['is_canceled']==0)]
for i in [city_hotel,resort_hotel]:
    i.index=range(i.shape[0])
```

In [9]:
```
city_month=city_hotel['arrival_date_month'].value_counts()
resort_month=resort_hotel['arrival_date_month'].value_counts()
name=resort_month.index
x=list(range(len(city_month.index)))
y=city_month.values
x1=[i+0.3 for i in x]
y1=resort_month.values
width=0.3
plt.figure(figsize=(15,8),dpi=80)
plt.plot(x,y,label='City Hotel',color='lightsalmon')
plt.plot(x1,y1,label='Resort Hotel',color='lightseagreen')
plt.xticks(x,name)
plt.legend()
```

```
plt.xlabel('Month')
plt.ylabel('Count')
plt.title('Month Book')
for x,y in zip(x,y):
    plt.text(x,y+0.1,'%d' % y,ha = 'center',va = 'bottom')

for x,y in zip(x1,y1):
    plt.text(x,y+0.1,'%d' % y,ha = 'center',va = 'bottom')
```



Comment: Peak booking months are August and July. Preliminary judgment is that the long holiday caused the peak period.

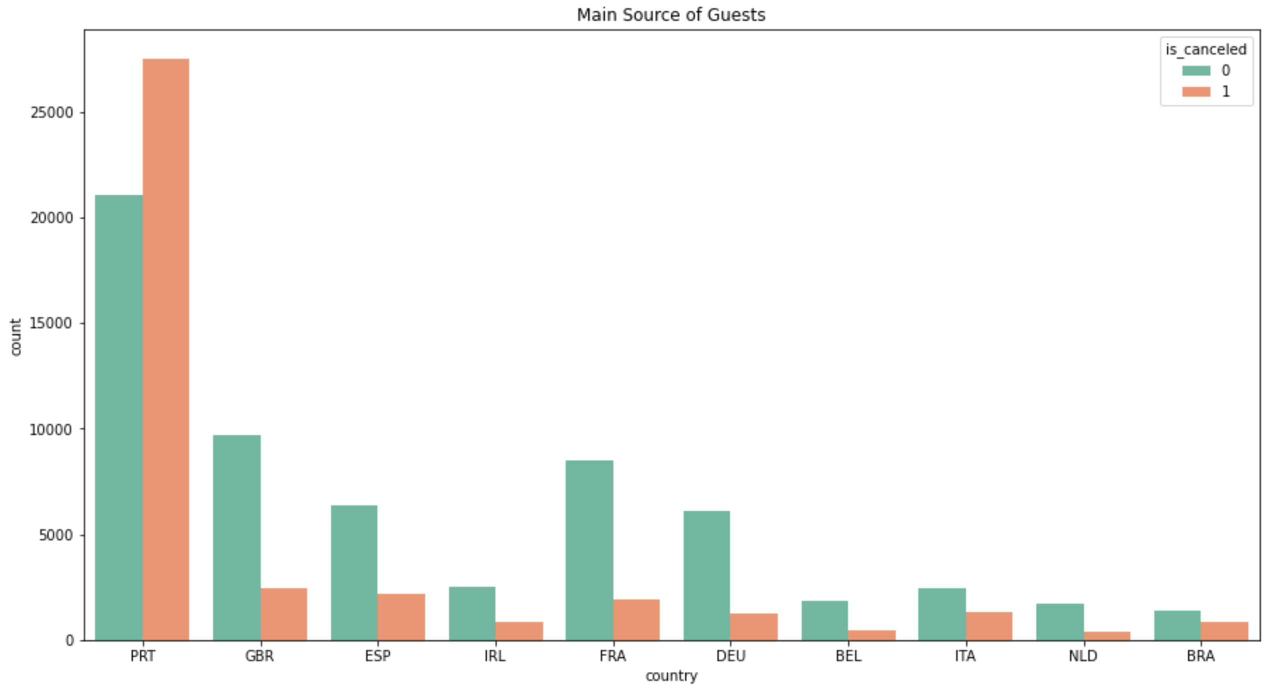
3. Customer origin and booking cancellation rate

In [10]:

```
#create a plot using Seaborn's countplot to visualize the top 10 countries from which bo
country_book=df['country'].value_counts()[:10]
country_cancel=df[(df.country.isin (country_book.index)) & (df.is_canceled==1)]['countr
plt.figure(figsize=(15,8))
sns.countplot(x='country'
              ,data=df[df.country.isin (country_book.index)]
              ,hue='is_canceled'
              ,palette=sns.color_palette('Set2',2)
             )
plt.title('Main Source of Guests')
```

Out[10]:

```
Text(0.5, 1.0, 'Main Source of Guests')
```



In [11]:

```
#calculate the cancellation rate for each of the top 10 countries (those with the highe
country_cancel_rate=(country_cancel/country_book).sort_values(ascending=False)
print('Customer cancellation rates by country'.center(10),country_cancel_rate,sep='\n')
```

```
Customer cancellation rates by country
PRT    0.566351
BRA    0.373201
ITA    0.353956
ESP    0.254085
IRL    0.246519
BEL    0.202391
GBR    0.202243
FRA    0.185694
NLD    0.183935
DEU    0.167147
Name: country, dtype: float64
```

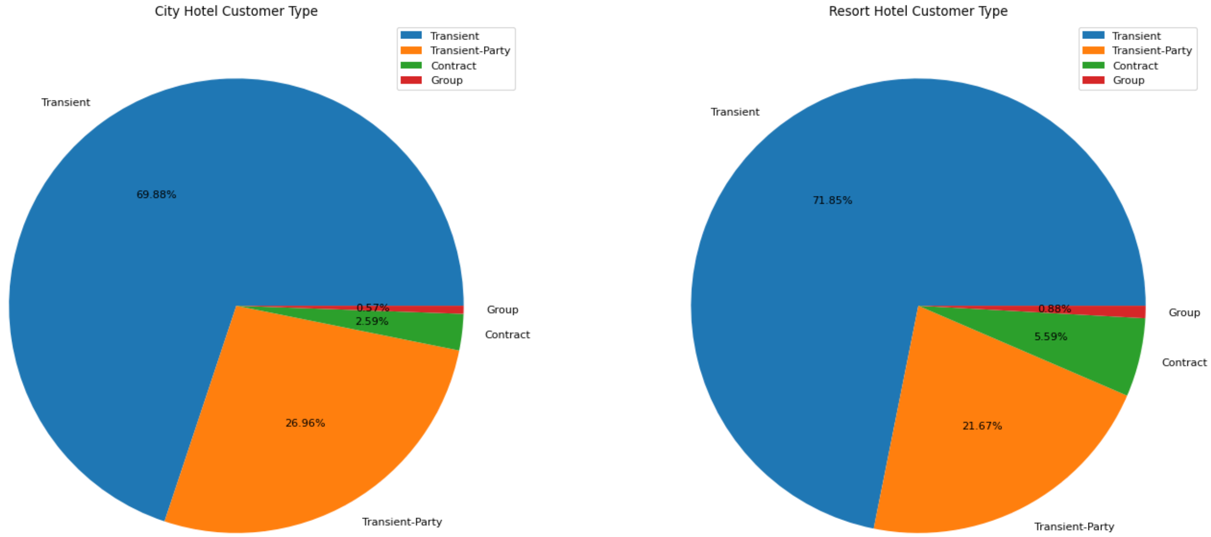
The peak season for both Resort hotel and City hotel is July and August in summer, and the main sources of tourists are European countries. This is in line with the characteristics of European tourists who prefer summer travel. It is necessary to focus on countries with high cancellation rates such as Portugal (PRT) and the United Kingdom (BRT). Main source of customers.

4. Customer type

In [12]:

```
#visualize the distribution of customer types for two types of hotels: City Hotel and Re
city_customer=city_hotel.customer_type.value_counts()
resort_customer=resort_hotel.customer_type.value_counts()
plt.figure(figsize=(21,12),dpi=80)
plt.subplot(1,2,1)
plt.pie(city_customer,labels=city_customer.index,autopct='%.2f%%')
plt.legend(loc=1)
plt.title('City Hotel Customer Type')
plt.subplot(1,2,2)
plt.pie(resort_customer,labels=resort_customer.index,autopct='%.2f%%')
```

```
plt.title('Resort Hotel Customer Type')
plt.legend()
plt.show()
```

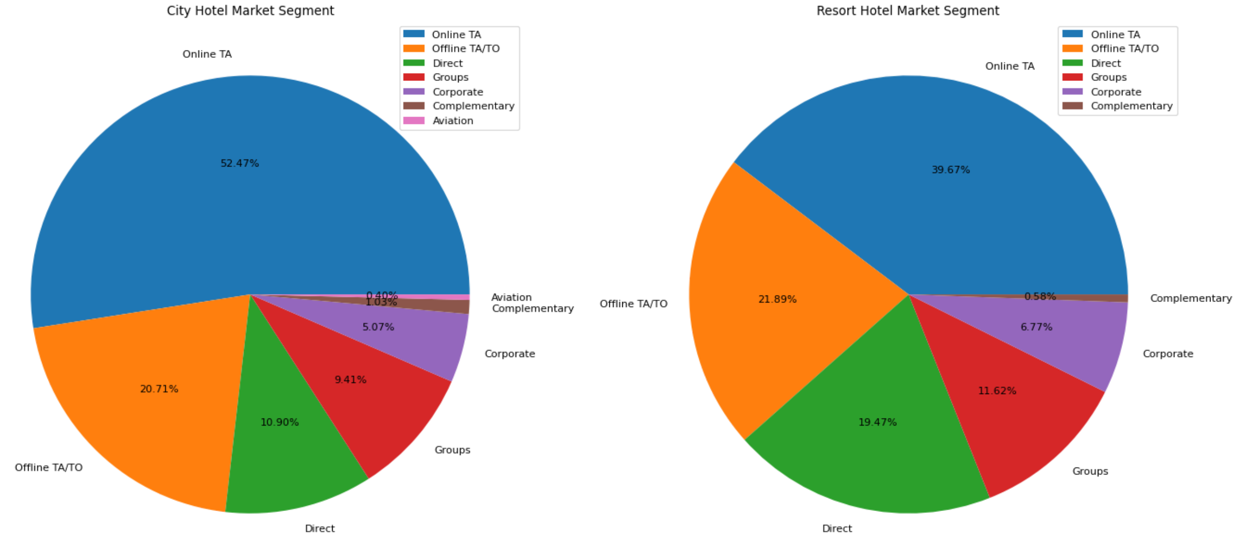


The main customer type of the hotel is transient travelers, accounting for about 70%.

5. Hotel booking method

In [13]:

```
#create pie charts to visualize the distribution of market segments for both City Hotel
city_segment=city_hotel.market_segment.value_counts()
resort_segment=resort_hotel.market_segment.value_counts()
plt.figure(figsize=(21,12),dpi=80)
plt.subplot(1,2,1)
plt.pie(city_segment,labels=city_segment.index,autopct='%.2f%%')
plt.legend()
plt.title('City Hotel Market Segment')
plt.subplot(1,2,2)
plt.pie(resort_segment,labels=resort_segment.index,autopct='%.2f%%')
plt.title('Resort Hotel Market Segment')
plt.legend()
plt.show()
```



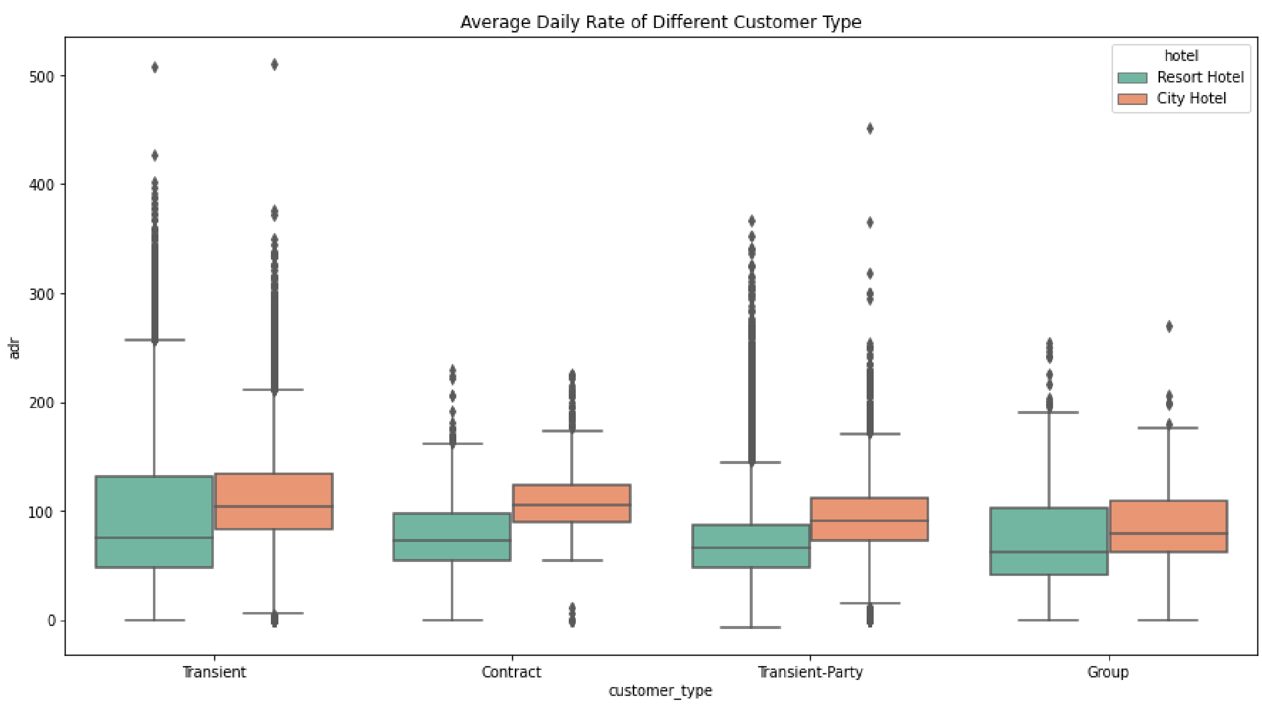
The customers of the two hotels mainly come from online travel agencies, which account for even more than 50% of the City Hotel; offline travel agencies come next, accounting for about 20%.

6. Average daily expenses of various types of passengers

In [14]:

```
#visualize the distribution of Average Daily Rate (adr) across different customer types,
plt.figure(figsize=(15,8))
sns.boxplot(x='customer_type'
            ,y='adr'
            ,hue='hotel'
            ,data=df[df.is_canceled==0]
            ,palette=sns.color_palette('Set2',2)
           )
plt.title('Average Daily Rate of Different Customer Type')
```

Out[14]: Text(0.5, 1.0, 'Average Daily Rate of Different Customer Type')



The average daily expenditure of all types of customers of City Hotel is higher than that of Resort Hotel; among the four types of customers, the consumption of individual travelers (Transient) is the highest and that of group travelers (Group) is the lowest.

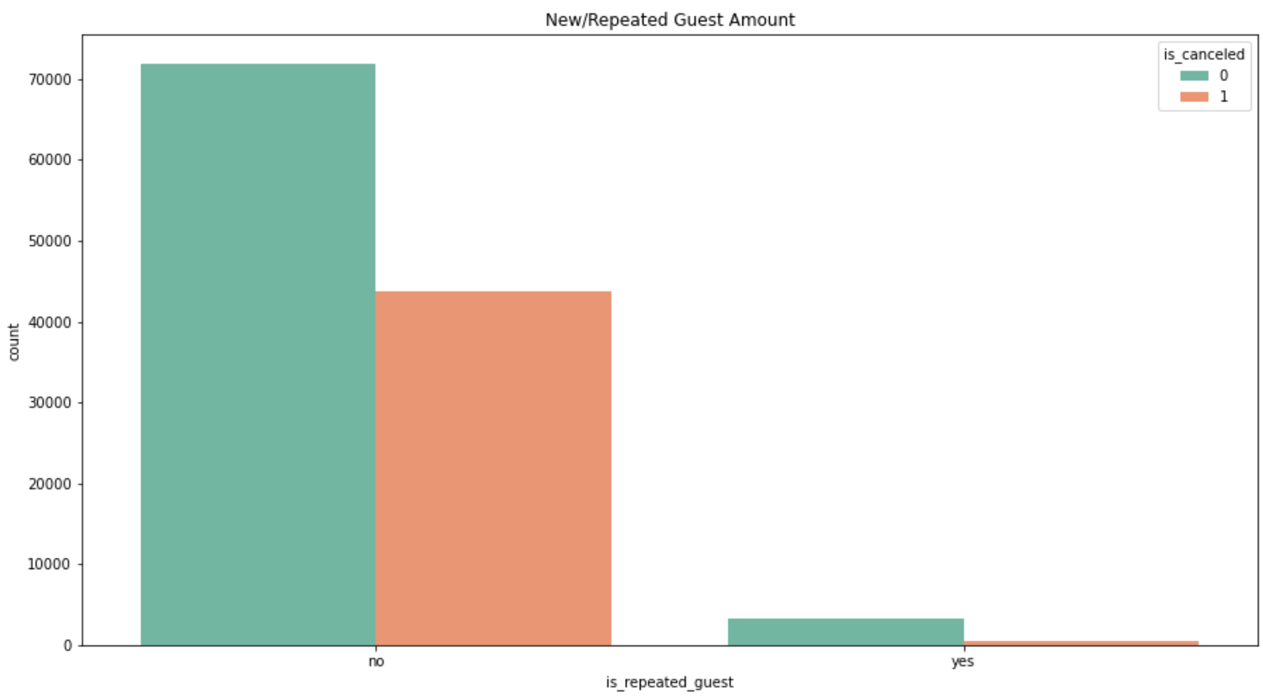
7.Number of new and old customers and cancellation rate

In [15]:

```
# visualize the count of bookings, categorized by whether the guest is a repeated guest
plt.figure(figsize=(15,8))
sns.countplot(x='is_repeated_guest'
              ,data=df
              ,hue='is_canceled'
              ,palette=sns.color_palette('Set2',2)
             )
plt.title('New/Repeated Guest Amount')
plt.xticks(range(2),['no','yes'])
```

Out[15]:

```
([<matplotlib.axis.XTick at 0x277488dff10>,
  <matplotlib.axis.XTick at 0x277488dfee0>],
 [Text(0, 0, 'no'), Text(1, 0, 'yes')])
```



In [16]:

```
#calculate and printing the cancellation rates for new and repeated guests
guest_cancel=(df.loc[df['is_canceled']==1]['is_repeated_guest'].value_counts()/df['is_r
guest_cancel.index=['New Guest', 'Repeated Guest']
print('Cancellation rate for new and old customers'.center(15),guest_cancel,sep='\n')
```

```
Cancellation rate for new and old customers
New Guest         0.377851
Repeated Guest    0.144882
Name: is_repeated_guest, dtype: float64
```

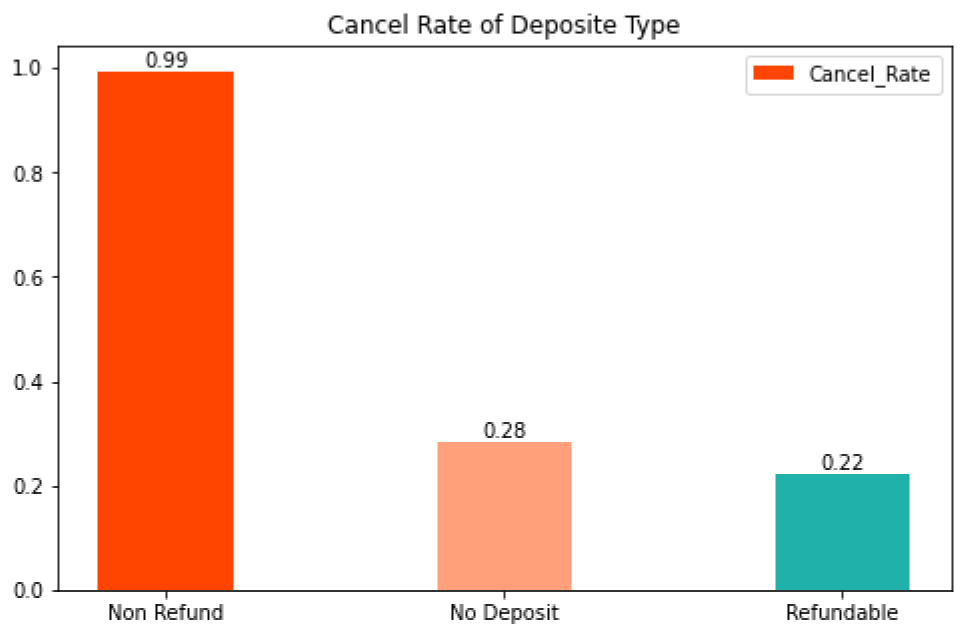
The cancellation rate for regular customers was 14.4%, while the cancellation rate for new customers reached 37.8%, which was 24 percentage points higher than that for regular customers.

8. Deposit method and reservation cancellation rate

In [17]:
```
print('Three deposit methods for booking quantity'.center(15),df['deposit_type'].value_
```

```
Three deposit methods for booking quantity
No Deposit    104641
Non Refund     14587
Refundable       162
Name: deposit_type, dtype: int64
```

In [18]:
```
#calculate the cancellation rates based on the 'deposit_type', and visualizing these ra
deposit_cancel=(df.loc[df['is_canceled']==1]['deposit_type'].value_counts()/df['deposit_
plt.figure(figsize=(8,5))
x=range(len(deposit_cancel.index))
y=deposit_cancel.values
plt.bar(x,y,label='Cancel_Rate',color=['orangered','lightsalmon','lightseagreen'],width
plt.xticks(x,deposit_cancel.index)
plt.legend()
plt.title('Cancel Rate of Deposite Type')
for x,y in zip(x,y):
    plt.text(x,y,'%.2f' % y,ha = 'center',va = 'bottom')
```



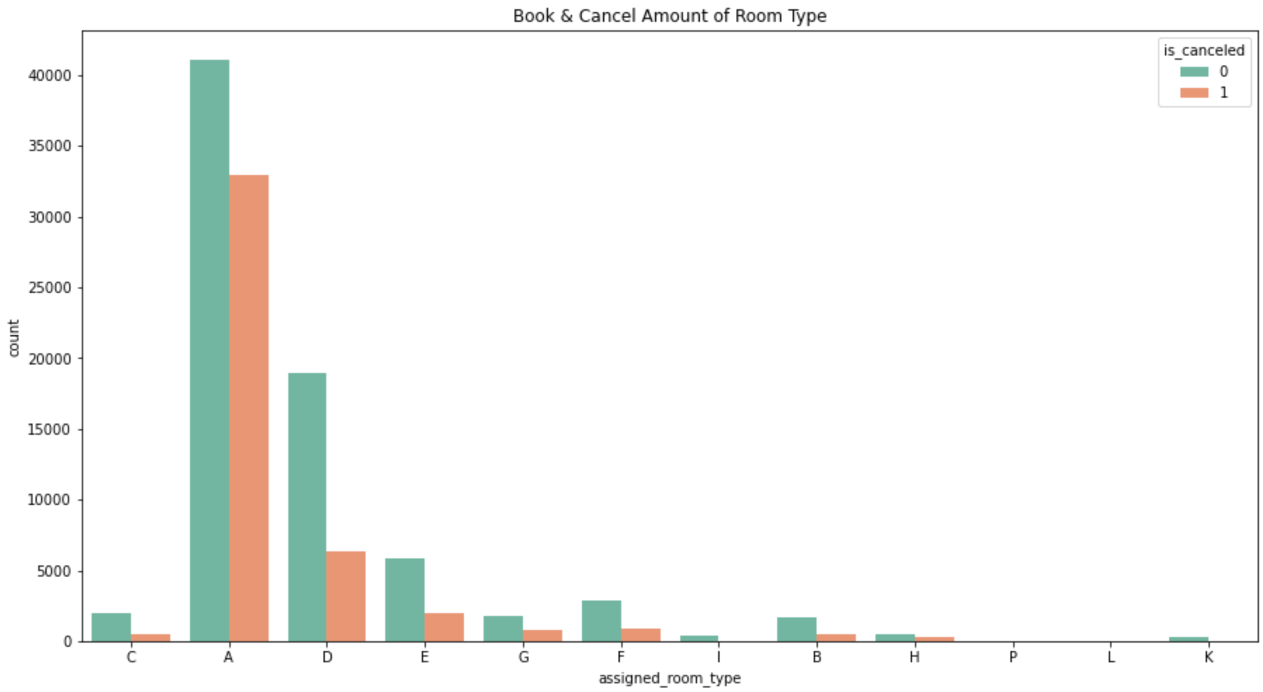
'No Deposit' is the method with the highest number of bookings and has a low cancellation rate, while the cancellation rate of non-refundable type is as high as 99%. This type of deposit method can be reduced to reduce Customer cancellation rate.

9. Room type and cancellation volume

In [19]:
```
#visualize the counts of bookings and cancellations across different assigned room type:
plt.figure(figsize=(15,8))
sns.countplot(x='assigned_room_type'
              ,data=df
              ,hue='is_canceled'
              ,palette=sns.color_palette('Set2',2)
```

```
                )
plt.title('Book & Cancel Amount of Room Type')
```

Out[19]: Text(0.5, 1.0, 'Book & Cancel Amount of Room Type')



In [20]:
```
room_cancel=df.loc[df['is_canceled']==1]['assigned_room_type'].value_counts()[:7]/df['a
print('Cancellation rates for different room types'.center(5),room_cancel.sort_values(a
```

```
Cancellation rates for different room types
A    0.444925
G    0.305523
E    0.252114
D    0.251244
F    0.247134
B    0.236708
C    0.187789
Name: assigned_room_type, dtype: float64
```

Among the top seven room types with the most bookings, the cancellation rates of room types A and G are higher than other room types, and the cancellation rate of room type A is as high as 44.5%.

10. Conclusion

1. The booking volume and cancellation rate of City Hotel are much higher than that of Resort Hotel. The hotel should conduct customer surveys to gain an in-depth understanding of the factors that cause customers to give up on bookings in order to reduce customer cancellation rates.
2. Hotels should make good use of the peak tourist season of July and August every year. They can increase prices appropriately while ensuring service quality to obtain more profits, and

conduct preferential activities during the off-season (winter), such as Christmas sales and New Year activities, to reduce Hotel vacancy rate.

3. Hotels need to analyze customer profiles from major source countries such as Portugal and the United Kingdom, understand the attribute tags, preferences and consumption characteristics of these customers, and launch exclusive services to reduce customer cancellation rates.

4. Since individual travelers are the main customer group of hotels and have high consumption levels, hotels can increase the promotion and marketing of independent travelers through online and offline travel agencies, thereby attracting more tourists of this type.

5. The cancellation rate of new customers is 24% higher than that of old customers. Therefore, hotels should focus on the booking and check-in experience of new customers, and provide more guidance and benefits to new customers, such as providing discounts to first-time customers and conducting research on new customers. Provide feedback on satisfaction and dissatisfaction with your stay to improve future services and maintain good old customers.

6. The cancellation rate of non-refundable deposits is as high as 99%. Hotels should optimize this method, such as returning 50% of the deposit, or cancel this method directly to increase the occupancy rate.

7. The cancellation rate of room types A and G is much higher than that of other room types. The hotel should carefully confirm the room information with the customer when making a reservation, so that the customer can fully understand the room situation, avoid cognitive errors, and at the same time be able to understand the room facilities. Optimize and improve service levels.

Data Processing (Week 4)

The purpose of modeling is to predict whether a passenger will cancel a hotel reservation, and then set 'is_canceled' to label y, and the rest to feature x. The date feature 'is_cance'reservation_status_date' does not directly affect the label, so remove it

In [21]:

```
#Create a new DataFrame 'df1' from 'df'
df1=df.drop(labels=['reservation_status_date'],axis=1)
```

Handling Categorical Variables

In [22]:

```
#Getting the names of all columns in 'df1'
cate=df1.columns[df1.dtypes == "object"].tolist()
#Categorical variables expressed as numbers
num_cate=['agent','company','is_repeated_guest']
cate=cate+num_cate
```

In [23]:

```
import numpy as np #linear algebra
#creating a dictionary
results={}
for i in ['agent','company']:
```

```
    result=np.sort(df1[i].unique())
    results[i]=result
results
```

Out[23]:

```
{'agent': array([  1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  10.,  11.,
         12.,  13.,  14.,  15.,  16.,  17.,  19.,  20.,  21.,  22.,  23.,
         24.,  25.,  26.,  27.,  28.,  29.,  30.,  31.,  32.,  33.,  34.,
         35.,  36.,  37.,  38.,  39.,  40.,  41.,  42.,  44.,  45.,  47.,
         50.,  52.,  53.,  54.,  55.,  56.,  57.,  58.,  59.,  60.,  61.,
         63.,  64.,  66.,  67.,  68.,  69.,  70.,  71.,  72.,  73.,  74.,
         75.,  77.,  78.,  79.,  81.,  82.,  83.,  85.,  86.,  87.,  88.,
         89.,  90.,  91.,  92.,  93.,  94.,  95.,  96.,  98.,  99., 103.,
        104., 105., 106., 107., 110., 111., 112., 114., 115., 117., 118.,
        119., 121., 122., 126., 127., 128., 129., 132., 133., 134., 135.,
        138., 139., 141., 142., 143., 144., 146., 147., 148., 149., 150.,
        151., 152., 153., 154., 155., 156., 157., 158., 159., 162., 163.,
        165., 167., 168., 170., 171., 173., 174., 175., 177., 179., 180.,
        181., 182., 183., 184., 185., 187., 191., 192., 193., 195., 196.,
        197., 201., 205., 208., 210., 211., 213., 214., 215., 216., 219.,
        220., 223., 227., 229., 232., 234., 235., 236., 240., 241., 242.,
        243., 244., 245., 247., 248., 249., 250., 251., 252., 253., 254.,
        256., 257., 258., 261., 262., 265., 267., 269., 270., 273., 275.,
        276., 278., 280., 281., 282., 283., 285., 286., 287., 288., 289.,
        290., 291., 294., 295., 296., 298., 299., 300., 301., 302., 303.,
        304., 305., 306., 307., 308., 310., 313., 314., 315., 321., 323.,
        324., 325., 326., 327., 328., 330., 331., 332., 333., 334., 335.,
        336., 337., 339., 341., 344., 346., 348., 350., 352., 354., 355.,
        358., 359., 360., 363., 364., 367., 368., 370., 371., 375., 378.,
        384., 385., 387., 388., 390., 391., 393., 394., 397., 403., 404.,
        405., 406., 408., 410., 411., 414., 416., 418., 420., 423., 425.,
        426., 427., 429., 430., 431., 432., 433., 434., 436., 438., 440.,
        441., 444., 446., 449., 450., 451., 453., 454., 455., 459., 461.,
        464., 467., 468., 469., 472., 474., 475., 476., 479., 480., 481.,
        483., 484., 492., 493., 495., 497., 502., 508., 509., 510., 526.,
        527., 531., 535.,  nan]),
 'company': array([  6.,   8.,   9.,  10.,  11.,  12.,  14.,  16.,  18.,  20.,  22.,
         28.,  29.,  31.,  32.,  34.,  35.,  37.,  38.,  39.,  40.,  42.,
         43.,  45.,  46.,  47.,  48.,  49.,  51.,  52.,  53.,  54.,  59.,
         61.,  62.,  64.,  65.,  67.,  68.,  71.,  72.,  73.,  76.,  77.,
         78.,  80.,  81.,  82.,  83.,  84.,  85.,  86.,  88.,  91.,  92.,
         93.,  94.,  96.,  99., 100., 101., 102., 103., 104., 105., 106.,
        107., 108., 109., 110., 112., 113., 115., 116., 118., 120., 122.,
        126., 127., 130., 132., 135., 137., 139., 140., 142., 143., 144.,
        146., 148., 149., 150., 153., 154., 158., 159., 160., 163., 165.,
        167., 168., 169., 174., 178., 179., 180., 183., 184., 185., 186.,
        192., 193., 195., 197., 200., 202., 203., 204., 207., 209., 210.,
        212., 213., 215., 216., 217., 218., 219., 220., 221., 222., 223.,
        224., 225., 227., 229., 230., 232., 233., 234., 237., 238., 240.,
        242., 243., 245., 246., 250., 251., 253., 254., 255., 257., 258.,
        259., 260., 263., 264., 268., 269., 270., 271., 272., 273., 274.,
        275., 277., 278., 279., 280., 281., 282., 284., 286., 287., 288.,
        289., 290., 291., 292., 293., 297., 301., 302., 304., 305., 307.,
        308., 309., 311., 312., 313., 314., 316., 317., 318., 319., 320.,
        321., 323., 324., 325., 329., 330., 331., 332., 333., 334., 337.,
        338., 341., 342., 343., 346., 347., 348., 349., 350., 351., 352.,
        353., 355., 356., 357., 358., 360., 361., 362., 364., 365., 366.,
        367., 368., 369., 370., 371., 372., 373., 376., 377., 378., 379.,
        380., 382., 383., 384., 385., 386., 388., 390., 391., 392., 393.,
        394., 395., 396., 397., 398., 399., 400., 401., 402., 403., 405.,
        407., 408., 409., 410., 411., 412., 413., 415., 416., 417., 418.,
```

```
       419., 420., 421., 422., 423., 424., 425., 426., 428., 429., 433.,
       435., 436., 437., 439., 442., 443., 444., 445., 446., 447., 448.,
       450., 451., 452., 454., 455., 456., 457., 458., 459., 460., 461.,
       465., 466., 470., 477., 478., 479., 481., 482., 483., 484., 485.,
       486., 487., 489., 490., 491., 492., 494., 496., 497., 498., 499.,
       501., 504., 506., 507., 511., 512., 513., 514., 515., 516., 518.,
       520., 521., 523., 525., 528., 530., 531., 534., 539., 541., 543.,
        nan])}
```

In [24]:

```
# the agent and company columns have a large number of empty values and no 0 values, so
df1[['agent','company']]=df1[['agent','company']].fillna(0,axis=0)
```

In [25]:

```
df1.loc[:,cate].isnull().mean()
```

Out[25]:

```
hotel                   0.000000
arrival_date_month      0.000000
meal                    0.000000
country                 0.004087
market_segment          0.000000
distribution_channel    0.000000
reserved_room_type      0.000000
assigned_room_type      0.000000
deposit_type            0.000000
customer_type           0.000000
reservation_status      0.000000
name                    0.000000
email                   0.000000
phone-number            0.000000
credit_card             0.000000
agent                   0.000000
company                 0.000000
is_repeated_guest       0.000000
dtype: float64
```

In [26]:

```
#create new variables in_company and in_agent to classify passengers. If company and age
df1.loc[df1['company'] == 0,'in_company']='NO'
df1.loc[df1['company'] != 0,'in_company']='YES'
df1.loc[df1['agent'] == 0,'in_agent']='NO'
df1.loc[df1['agent'] != 0,'in_agent']='YES'
```

In [27]:

```
#create a new feature same_assignment. If the booked room type is consistent with the a
df1.loc[df1['reserved_room_type'] == df1['assigned_room_type'],'same_assignment']='Yes'
df1.loc[df1['reserved_room_type'] != df1['assigned_room_type'],'same_assignment']='No'
```

In [28]:

```
#delete four features except 'reserved_room_type', 'assigned_room_type', 'agent', 'compa
df1=df1.drop(labels=['reserved_room_type','assigned_room_type','agent','company'],axis=
```

In [29]:

```
#reset 'is_repeated_guest', frequent guests are marked as YES, non-repeated guests are n
df1['is_repeated_guest'][df1['is_repeated_guest']==0]='NO'
df1['is_repeated_guest'][df1['is_repeated_guest']==1]='YES'
```

In [30]:
```
#filling the missing values in the 'country' column of the DataFrame 'df1' with the mode
df1['country']=df1['country'].fillna(df1['country'].mode()[0])
```

In [31]:
```
for i in ['in_company','in_agent','same_assignment']:
    cate.append(i)

for i in ['reserved_room_type','assigned_room_type','agent','company']:
    cate.remove(i)
cate
```

Out[31]:
```
['hotel',
 'arrival_date_month',
 'meal',
 'country',
 'market_segment',
 'distribution_channel',
 'deposit_type',
 'customer_type',
 'reservation_status',
 'name',
 'email',
 'phone-number',
 'credit_card',
 'is_repeated_guest',
 'in_company',
 'in_agent',
 'same_assignment']
```

In [32]:
```
#encoding categorical features
from sklearn.preprocessing import OrdinalEncoder
oe = OrdinalEncoder()
oe = oe.fit(df1.loc[:,cate])
df1.loc[:,cate] = oe.transform(df1.loc[:,cate])
```

Working With Continuous Variables

In [33]:
```
#to filter out continuous variables, you need to delete the label 'is_canceled' first.
col=df1.columns.tolist()
col.remove('is_canceled')
for i in cate:
    col.remove(i)
col
```

Out[33]:
```
['lead_time',
 'arrival_date_year',
 'arrival_date_week_number',
 'arrival_date_day_of_month',
 'stays_in_weekend_nights',
 'stays_in_week_nights',
 'adults',
 'children',
 'babies',
 'previous_cancellations',
 'previous_bookings_not_canceled',
 'booking_changes',
```

```
'days_in_waiting_list',
 'adr',
 'required_car_parking_spaces',
 'total_of_special_requests']
```

In [34]:
```
df1[col].isnull().sum()
```

Out[34]:
```
lead_time                         0
arrival_date_year                 0
arrival_date_week_number          0
arrival_date_day_of_month         0
stays_in_weekend_nights           0
stays_in_week_nights              0
adults                            0
children                          4
babies                            0
previous_cancellations            0
previous_bookings_not_canceled    0
booking_changes                   0
days_in_waiting_list              0
adr                               0
required_car_parking_spaces       0
total_of_special_requests         0
dtype: int64
```

In [35]:
```
#use mode to fill null values in xtrain children column
df1['children']=df1['children'].fillna(df1['children'].mode()[0])
```

In [36]:
```
#continuous variables are dimensionless
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
ss = ss.fit(df1.loc[:,col])
df1.loc[:,col] = ss.transform(df1.loc[:,col])
```

Correlation Coefficient of Each Variable

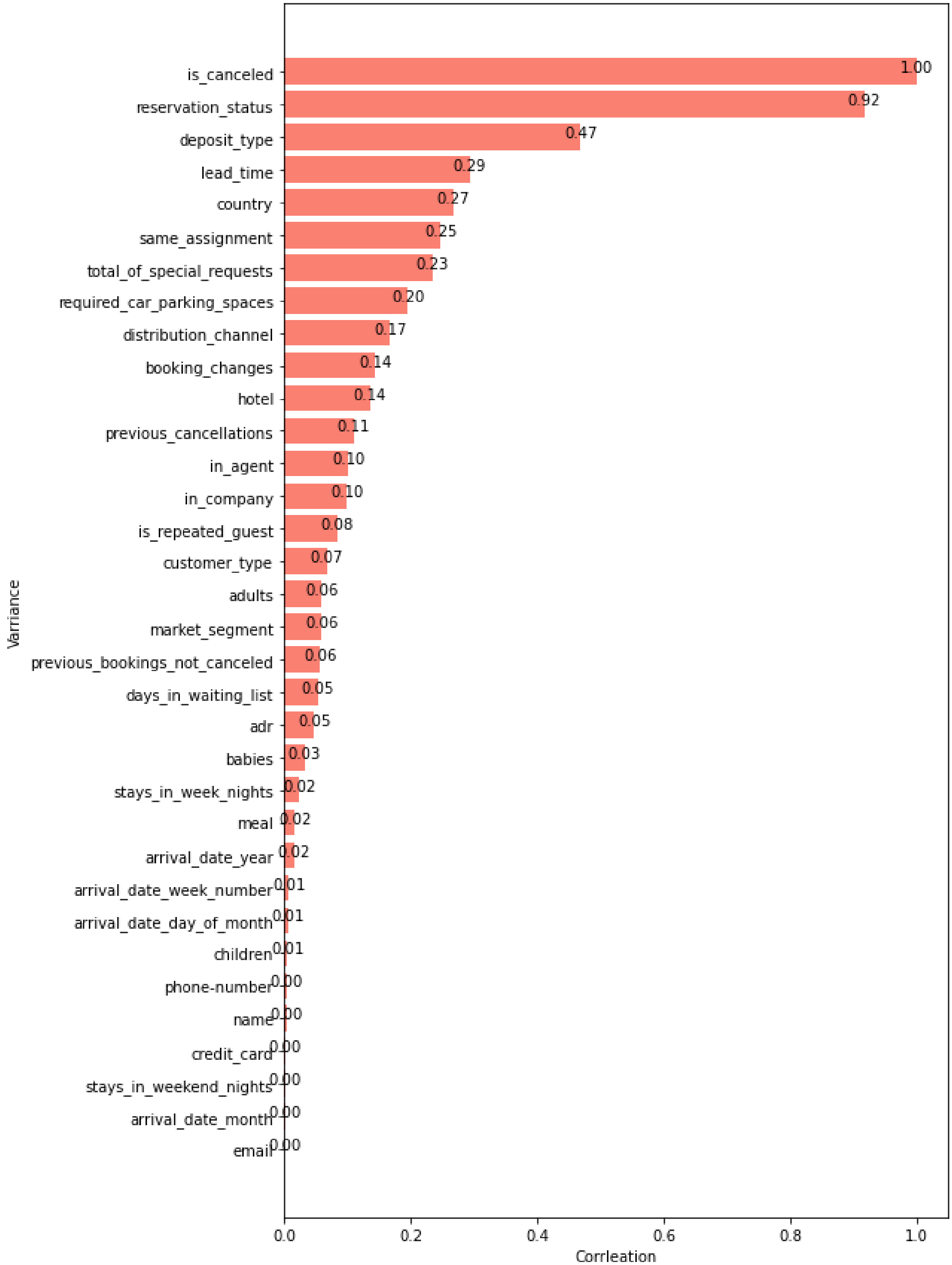
In [37]:
```
#calculating the correlation of all numerical columns with the 'is_canceled column' in
cor=df1.corr()
cor=abs(cor['is_canceled']).sort_values()
cor
```

Out[37]:
```
email                             0.000723
arrival_date_month                0.001491
stays_in_weekend_nights           0.001791
credit_card                       0.002515
name                              0.004253
phone-number                      0.004342
children                          0.005036
arrival_date_day_of_month         0.006130
arrival_date_week_number          0.008148
arrival_date_year                 0.016660
meal                              0.017678
stays_in_week_nights              0.024765
babies                            0.032491
adr                               0.047557
```

```
days_in_waiting_list              0.054186
previous_bookings_not_canceled    0.057358
market_segment                    0.059338
adults                            0.060017
customer_type                     0.068140
is_repeated_guest                 0.084793
in_company                        0.099310
in_agent                          0.102068
previous_cancellations            0.110133
hotel                             0.136531
booking_changes                   0.144381
distribution_channel              0.167600
required_car_parking_spaces       0.195498
total_of_special_requests         0.234658
same_assignment                   0.247770
country                           0.267502
lead_time                         0.293123
deposit_type                      0.468634
reservation_status                0.917196
is_canceled                       1.000000
Name: is_canceled, dtype: float64
```

In [38]:

```
#create a horizontal bar plot using Matplotlib to visualize the absolute correlation va
plt.figure(figsize=(8,15))
x=range(len(cor.index))
name=cor.index
y=abs(cor.values)
plt.barh(x,y,color='salmon')
plt.yticks(x,name)
for x,y in zip(x,y):
    plt.text(y,x-0.1,'%.2f' % y,ha = 'center',va = 'bottom')
plt.xlabel('Corrleation')
plt.ylabel('Varriance')
plt.show()
```



The reservation status ('reservation_status') has the highest correlation with whether to cancel the reservation, reaching 0.92, but considering that it may cause the model to overfit in the future, it is deleted; the deposit type ('deposit_type') reaches 0.47, creating a characteristic Whether the reservation and assigned room type are consistent ('same_assignment') also has a correlation of 0.25.

In [39]:
```
#copy 'df1' with the column labeled 'reservation_status' dropped.
df2=df1.drop('reservation_status',axis=1)
```

Week 5

In [40]:
```
df2.head()
```

Out[40]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_
0	1.0	0	2.227051	-1.634768	5.0	-0.012141	
1	1.0	0	5.923385	-1.634768	5.0	-0.012141	
2	1.0	0	-0.907814	-1.634768	5.0	-0.012141	
3	1.0	0	-0.851667	-1.634768	5.0	-0.012141	
4	1.0	0	-0.842309	-1.634768	5.0	-0.012141	

5 rows × 33 columns

In [41]:
```
df2.var()
```

Out[41]:
```
hotel                             2.229544e-01
is_canceled                       2.332100e-01
lead_time                         1.000008e+00
arrival_date_year                 1.000008e+00
arrival_date_month                1.249675e+01
arrival_date_week_number          1.000008e+00
arrival_date_day_of_month         1.000008e+00
stays_in_weekend_nights           1.000008e+00
stays_in_week_nights              1.000008e+00
adults                            1.000008e+00
children                          1.000008e+00
babies                            1.000008e+00
meal                              1.141902e+00
country                           1.995000e+03
market_segment                    1.604594e+00
distribution_channel              8.236978e-01
is_repeated_guest                 3.089409e-02
previous_cancellations            1.000008e+00
previous_bookings_not_canceled    1.000008e+00
booking_changes                   1.000008e+00
deposit_type                      1.120096e-01
days_in_waiting_list              1.000008e+00
customer_type                     3.329753e-01
adr                               1.000008e+00
required_car_parking_spaces       1.000008e+00
total_of_special_requests         1.000008e+00
name                              5.369325e+08
email                             1.114750e+09
phone-number                      1.187841e+09
```

```
credit_card                    6.748231e+06
in_company                     5.369037e-02
in_agent                       1.181321e-01
same_assignment                1.093335e-01
dtype: float64
```

In [42]:

```
#divide feature x and label y
x=df2.loc[:,df2.columns != 'is_canceled' ]
y=df2.loc[:,'is_canceled']
from sklearn.model_selection import train_test_split as tts
xtrain,xtest,ytrain,ytest=tts(x,y,test_size=0.3,random_state=90)
for i in [xtrain,xtest,ytrain,ytest]:
    i.index=range(i.shape[0])
```

In [43]:

```
x.shape, y.shape
```

Out[43]:

```
((119390, 32), (119390,))
```

In [44]:

```
xtrain.head()
```

Out[44]:

	hotel	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_
0	0.0	1.029252	1.192195	5.0	0.061361	-0.5
1	0.0	0.102829	-0.221286	8.0	-0.600156	-1.5
2	1.0	0.168334	1.192195	6.0	-0.085642	1.6
3	1.0	0.767233	1.192195	5.0	-0.012141	-0.8
4	0.0	-0.421208	-0.221286	11.0	0.943385	1.1

5 rows × 32 columns

In [45]:

```
xtest.head()
```

Out[45]:

	hotel	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_
0	0.0	-0.963961	-1.634768	2.0	1.898910	1.2
1	0.0	-0.861025	-0.221286	5.0	0.208365	0.3
2	0.0	1.431638	-0.221286	5.0	0.355369	1.7
3	0.0	-0.879741	-0.221286	11.0	0.722879	-1.2
4	0.0	-0.224694	-0.221286	11.0	0.943385	1.1

5 rows × 32 columns

In [46]:
```
ytrain.head(), ytest.head()
```

Out[46]:
```
(0    1
 1    0
 2    0
 3    1
 4    0
 Name: is_canceled, dtype: int64,
 0    0
 1    0
 2    0
 3    0
 4    0
 Name: is_canceled, dtype: int64)
```

Week 6(XGBoost)

In [47]:
```
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score
from sklearn.model_selection import train_test_split


# Define the models with different hyperparameters
XGmodel_variations = {
    'XGmodel_1': XGBClassifier(n_estimators=100, learning_rate=0.1, random_state=90),
    'XGmodel_2': XGBClassifier(n_estimators=100, learning_rate=0.01, random_state=90),
    'XGmodel_3': XGBClassifier(n_estimators=100, learning_rate=0.1, max_depth=3, random_
}

# Dictionary to store the performance metrics for each model
performance_metrics = {}
```

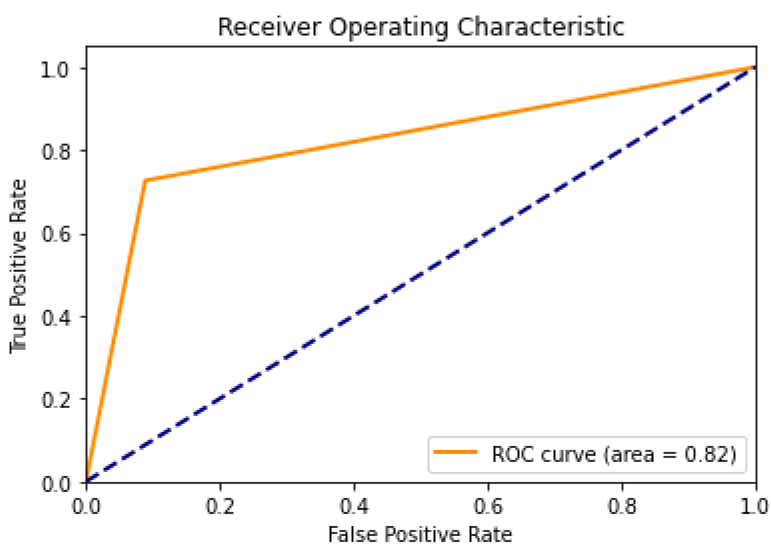
In [48]:
```
# Train each model and calculate metrics
for name, model in XGmodel_variations.items():
    model.fit(xtrain, ytrain)
    ytrain_pred = model.predict(xtrain)
    ytest_pred = model.predict(xtest)

    performance_metrics[name] = {
        'train_accuracy': accuracy_score(ytrain, ytrain_pred),
        'train_f1': f1_score(ytrain, ytrain_pred),
        'train_roc_auc': roc_auc_score(ytrain, ytrain_pred),
        'test_accuracy': accuracy_score(ytest, ytest_pred),
        'test_f1': f1_score(ytest, ytest_pred),
        'test_roc_auc': roc_auc_score(ytest, ytest_pred)
    }
```

In [49]:
```
# Output the performance metrics
for model, metrics in performance_metrics.items():
    print(f"Metrics for {model}:")
    for metric_name, metric_value in metrics.items():
        print(f"{metric_name}: {metric_value}")
```

```
Metrics for XGmodel_1:
train_accuracy: 0.8655307336101372
train_f1: 0.8111514418229482
train_roc_auc: 0.8473398907978615
test_accuracy: 0.8603456459223274
test_f1: 0.8015079365079366
test_roc_auc: 0.8411108612062882
Metrics for XGmodel_2:
train_accuracy: 0.8131094970863796
train_f1: 0.7025688877039971
train_roc_auc: 0.7683183623199193
test_accuracy: 0.8147248513275819
test_f1: 0.7020206555904804
test_roc_auc: 0.7685136787353957
Metrics for XGmodel_3:
train_accuracy: 0.8441123329304919
train_f1: 0.776189658134341
train_roc_auc: 0.8201658933607971
test_accuracy: 0.8434263059441047
test_f1: 0.7727346409466689
test_roc_auc: 0.8186964422810334
```

In [50]:

```python
from sklearn.metrics import roc_curve, auc

# Compute ROC curve and ROC area for each class
fpr, tpr, _ = roc_curve(ytest, ytest_pred)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```



Week 7 (Logistic)

In [51]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection import train_test_split

# Define the models with different hyperparameters
LGmodel_variations = {
    'LGmodel_1': LogisticRegression(C=1.0, max_iter=100, solver='lbfgs', random_state=9
    'LGmodel_2': LogisticRegression(C=0.5, max_iter=100, solver='lbfgs', random_state=9
    'LGmodel_3': LogisticRegression(C=1.0, max_iter=200, solver='liblinear', random_sta
}

# Dictionary to store the performance metrics for each model
performance_metrics = {}
```

In [52]:
```
# Train each model and calculate metrics
for name, model in LGmodel_variations.items():
    model.fit(xtrain, ytrain)
    ytrain_pred = model.predict(xtrain)
    ytest_pred = model.predict(xtest)

    performance_metrics[name] = {
        'train_accuracy': accuracy_score(ytrain, ytrain_pred),
        'train_precision': precision_score(ytrain, ytrain_pred),
        'train_recall': recall_score(ytrain, ytrain_pred),
        'train_f1': f1_score(ytrain, ytrain_pred),
        'test_accuracy': accuracy_score(ytest, ytest_pred),
        'test_precision': precision_score(ytest, ytest_pred),
        'test_recall': recall_score(ytest, ytest_pred),
        'test_f1': f1_score(ytest, ytest_pred)
    }
```

In [53]:
```
# Output the performance metrics
for model, metrics in performance_metrics.items():
    print(f"Metrics for {model}:")
    for metric_name, metric_value in metrics.items():
        print(f"{metric_name}: {metric_value}")
```

```
Metrics for LGmodel_1:
train_accuracy: 0.6254292654326158
train_precision: 0.4935769857065316
train_recall: 0.263227300504969З
train_f1: 0.34334619902668234
test_accuracy: 0.6269648490940056
test_precision: 0.48432343234323433
test_recall: 0.2681793954161273
test_f1: 0.3452095074736584
Metrics for LGmodel_2:
train_accuracy: 0.6254292654326158
train_precision: 0.4935769857065316
train_recall: 0.263227300504969З
train_f1: 0.34334619902668234
test_accuracy: 0.6269648490940056
```

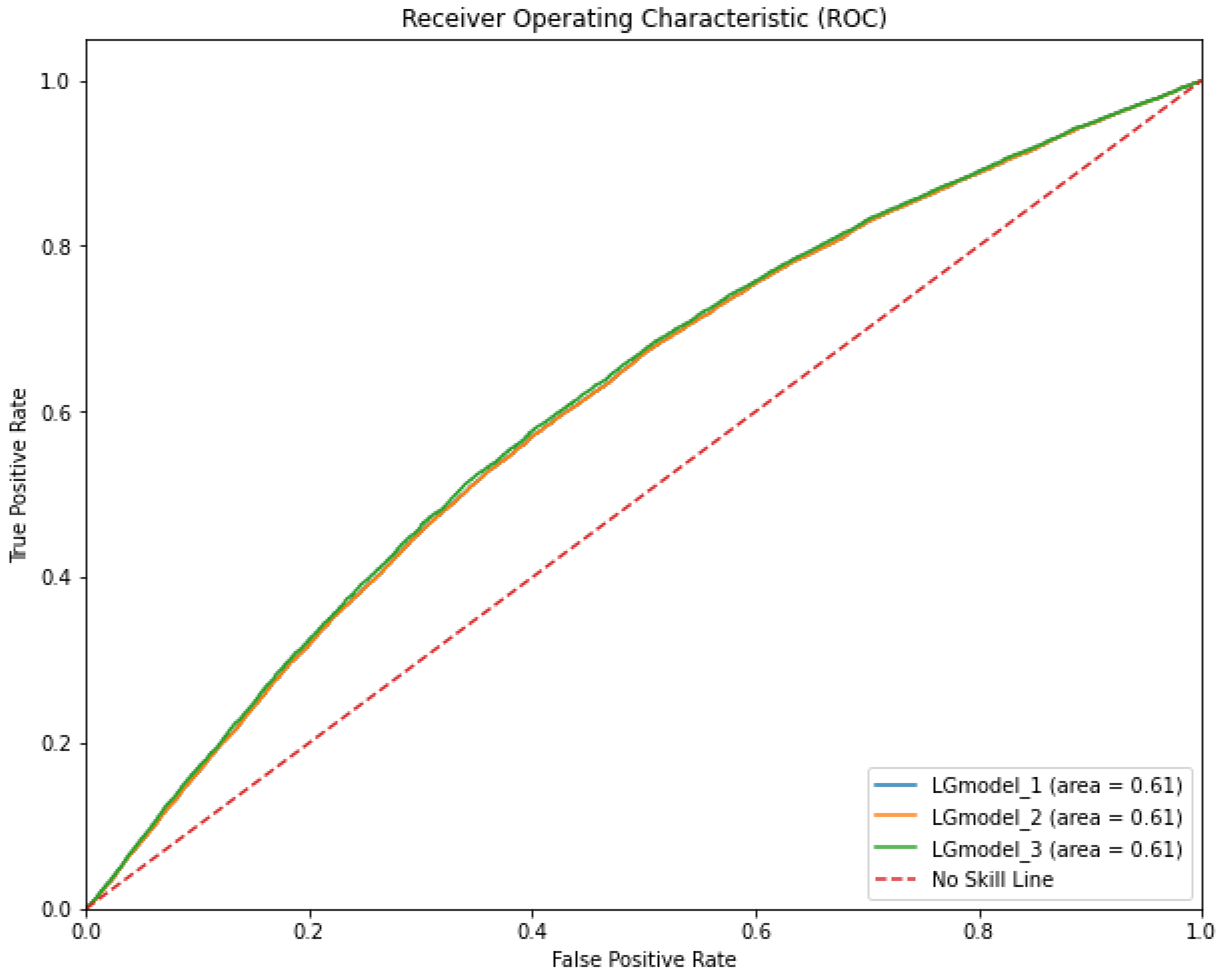
```
test_precision: 0.48432343234323433
test_recall: 0.2681793954161273
test_f1: 0.3452095074736584
Metrics for LGmodel_3:
train_accuracy: 0.6293061156115013
train_precision: 0.5035542747358309
train_recall: 0.25290276929014827
train_f1: 0.3367019226651822
test_accuracy: 0.6295334617639669
test_precision: 0.4900351699882767
test_recall: 0.25462575192263764
test_f1: 0.33512050909455326
```

In [54]:

```python
# Initialize a plt.figure to plot
plt.figure(figsize=(10, 8))

# Iterate over each model to calculate ROC curve and AUC
for name, model in LGmodel_variations.items():
    # Fit the model
    model.fit(xtrain, ytrain)

    # Get the prediction probabilities for the positive class
    ytest_proba = model.predict_proba(xtest)[:, 1]

    # Calculate ROC curve and ROC area
    fpr, tpr, _ = roc_curve(ytest, ytest_proba)
    roc_auc = auc(fpr, tpr)

    # Plot ROC curve
    plt.plot(fpr, tpr, label=f'{name} (area = {roc_auc:.2f})')

# Plot the No Skill Line
plt.plot([0, 1], [0, 1], linestyle='--', label='No Skill Line')

# Formatting the plot
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
```

Out[54]:

```
<matplotlib.legend.Legend at 0x2774a5c1490>
```



Week 8

In [55]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection import train_test_split

# Define the models with different hyperparameters
RFmodel_variations = {
    'RFmodel_1': RandomForestClassifier(n_estimators=100, max_depth=10, random_state=90
    'RFmodel_2': RandomForestClassifier(n_estimators=200, max_depth=15, min_samples_spl
    'RFmodel_3': RandomForestClassifier(n_estimators=300, max_depth=20, min_samples_spl
}

# Dictionary to store the performance metrics for each model
performance_metrics = {}
```

In [56]:

```
# Train each model and calculate metrics
for name, model in RFmodel_variations.items():
    model.fit(xtrain, ytrain)
    ytrain_pred = model.predict(xtrain)
    ytest_pred = model.predict(xtest)

    performance_metrics[name] = {
        'train_accuracy': accuracy_score(ytrain, ytrain_pred),
        'train_precision': precision_score(ytrain, ytrain_pred),
```

```
        'train_recall': recall_score(ytrain, ytrain_pred),
        'train_f1': f1_score(ytrain, ytrain_pred),
        'test_accuracy': accuracy_score(ytest, ytest_pred),
        'test_precision': precision_score(ytest, ytest_pred),
        'test_recall': recall_score(ytest, ytest_pred),
        'test_f1': f1_score(ytest, ytest_pred)
    }
```

In [57]:

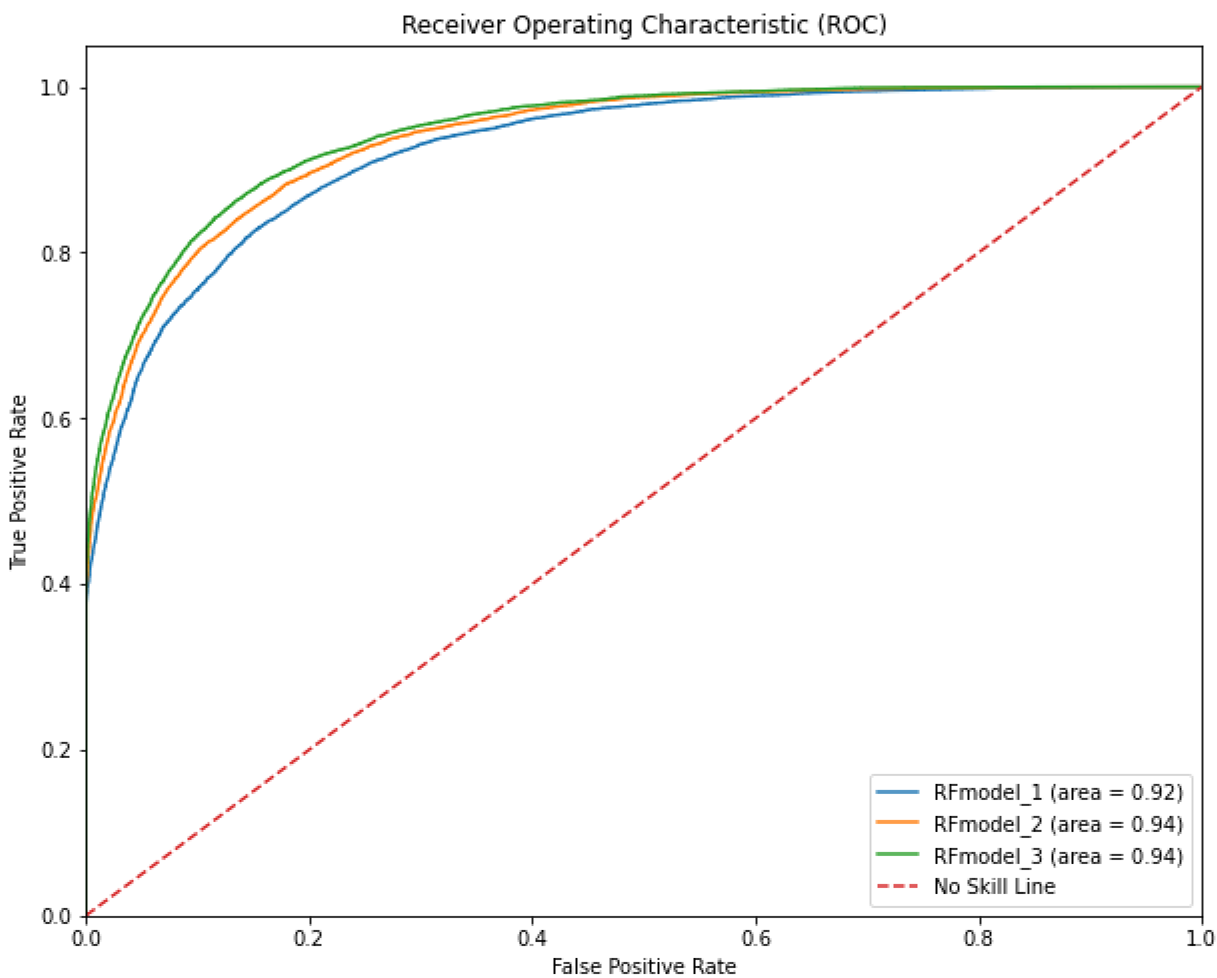
```
# Output the performance metrics
for model, metrics in performance_metrics.items():
    print(f"Metrics for {model}:")
    for metric_name, metric_value in metrics.items():
        print(f"{metric_name}: {metric_value}")
```

```
Metrics for RFmodel_1:
train_accuracy: 0.8485994280449427
train_precision: 0.8733597926453912
train_recall: 0.6936090830143772
train_f1: 0.773174623093057
test_accuracy: 0.8475305022754558
test_precision: 0.8677852348993289
test_recall: 0.6891799284245793
test_f1: 0.7682383397699785
Metrics for RFmodel_2:
train_accuracy: 0.8822107618489226
train_precision: 0.8883851862684167
train_recall: 0.7815766620565437
train_f1: 0.8315652590513998
test_accuracy: 0.8634168132451071
test_precision: 0.8633918334950172
test_recall: 0.7454503921419325
test_f1: 0.8000980712651193
Metrics for RFmodel_3:
train_accuracy: 0.9223074437916552
train_precision: 0.9242204746136865
train_recall: 0.8618249654240777
train_f1: 0.8919328262570112
test_accuracy: 0.8700058631376162
test_precision: 0.8651046601774486
test_recall: 0.7647148404781847
test_f1: 0.8118179613612482
```

In [58]:

```
# Initialize a plt.figure to plot
plt.figure(figsize=(10, 8))

# Iterate over each model to calculate ROC curve and AUC
for name, model in RFmodel_variations.items():
    # Fit the model
    model.fit(xtrain, ytrain)

    # Get the prediction probabilities for the positive class
    ytest_proba = model.predict_proba(xtest)[:, 1]

    # Calculate ROC curve and ROC area
    fpr, tpr, _ = roc_curve(ytest, ytest_proba)
    roc_auc = auc(fpr, tpr)

    # Plot ROC curve
```

```
    plt.plot(fpr, tpr, label=f'{name} (area = {roc_auc:.2f})')

# Plot the No Skill Line
plt.plot([0, 1], [0, 1], linestyle='--', label='No Skill Line')

# Formatting the plot
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")

# Show the plot
plt.show()
```



ANN

In [59]:

```
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score

# Assuming df2 is your dataframe after preprocessing and 'is_canceled' is the target va
x = df2.drop('is_canceled', axis=1).values  # Converting to NumPy array for Keras
y = df2['is_canceled'].values

# Split the data into training and testing sets
```

```
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.3, random_state=90)

# Define the hyperparameter settings for three variations
# Variation might include number of layers, number of neurons, activation functions, et
# vary the number of neurons in the first layer
hyperparameter_variations = [
    {'first_layer_neurons': 16},
    {'first_layer_neurons': 32},
    {'first_layer_neurons': 64}
]

# Define a function to create a model
def create_model(first_layer_neurons):
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(first_layer_neurons, activation='relu', input_shape=(xtra
        tf.keras.layers.Dense(1, activation='sigmoid')  # Output layer
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

In [60]:

```
# Train and evaluate each model variation
for i, params in enumerate(hyperparameter_variations, start=1):
    model = create_model(**params)
    model.fit(xtrain, ytrain, epochs=10, batch_size=32, verbose=0)  # You can adjust ep

    # Predict on training and validation set
    ytrain_pred = (model.predict(xtrain) > 0.5).astype("int32")
    ytest_pred = (model.predict(xtest) > 0.5).astype("int32")

    # Calculate performance metrics for both training and test datasets
    metrics = {
        'train_accuracy': accuracy_score(ytrain, ytrain_pred),
        'train_f1': f1_score(ytrain, ytrain_pred),
        'train_roc_auc': roc_auc_score(ytrain, ytrain_pred.ravel()),
        'test_accuracy': accuracy_score(ytest, ytest_pred),
        'test_f1': f1_score(ytest, ytest_pred),
        'test_roc_auc': roc_auc_score(ytest, ytest_pred.ravel())
    }

    # Print the performance metrics
    print(f"Model Variation {i}: {params}")
    print("Training Metrics:", metrics['train_accuracy'], metrics['train_f1'], metrics[
    print("Test Metrics:", metrics['test_accuracy'], metrics['test_f1'], metrics['test_
    print("-" * 30)
```

```
2612/2612 [==============================] - 3s 1ms/step
1120/1120 [==============================] - 1s 1ms/step
Model Variation 1: {'first_layer_neurons': 16}
Training Metrics: 0.7296255967836502 0.5243548183387362 0.6625719011709881
Test Metrics: 0.7321383700477427 0.5211619085645837 0.6616994905410234
------------------------------
2612/2612 [==============================] - 3s 1ms/step
1120/1120 [==============================] - 1s 1ms/step
Model Variation 2: {'first_layer_neurons': 32}
Training Metrics: 0.7161164490924102 0.46462822971905676 0.6376575759394365
Test Metrics: 0.7181505988776279 0.46018929469012354 0.635940865520492
------------------------------
2612/2612 [==============================] - 4s 1ms/step
1120/1120 [==============================] - 2s 1ms/step
```

```
Model Variation 3: {'first_layer_neurons': 64}
Training Metrics: 0.5574288346714849 0.5929724554588373 0.6204263908462381
Test Metrics: 0.5549878549292235 0.5876173967038368 0.6201876491319637
------------------------------
```

Week 9

In [61]:
```
# Split the training set into a smaller training set and a validation set
xtrain_new, xval, ytrain_new, yval = train_test_split(xtrain, ytrain, test_size=0.2, ra
```

In [62]:
```
from sklearn.metrics import roc_auc_score
# Evaluate all models using the validation dataset
validation_scores = {}
for name, model in {**XGmodel_variations, **LGmodel_variations, **RFmodel_variations}.i
    yval_pred = model.predict_proba(xval)[:, 1]
    validation_scores[name] = roc_auc_score(yval, yval_pred)
```

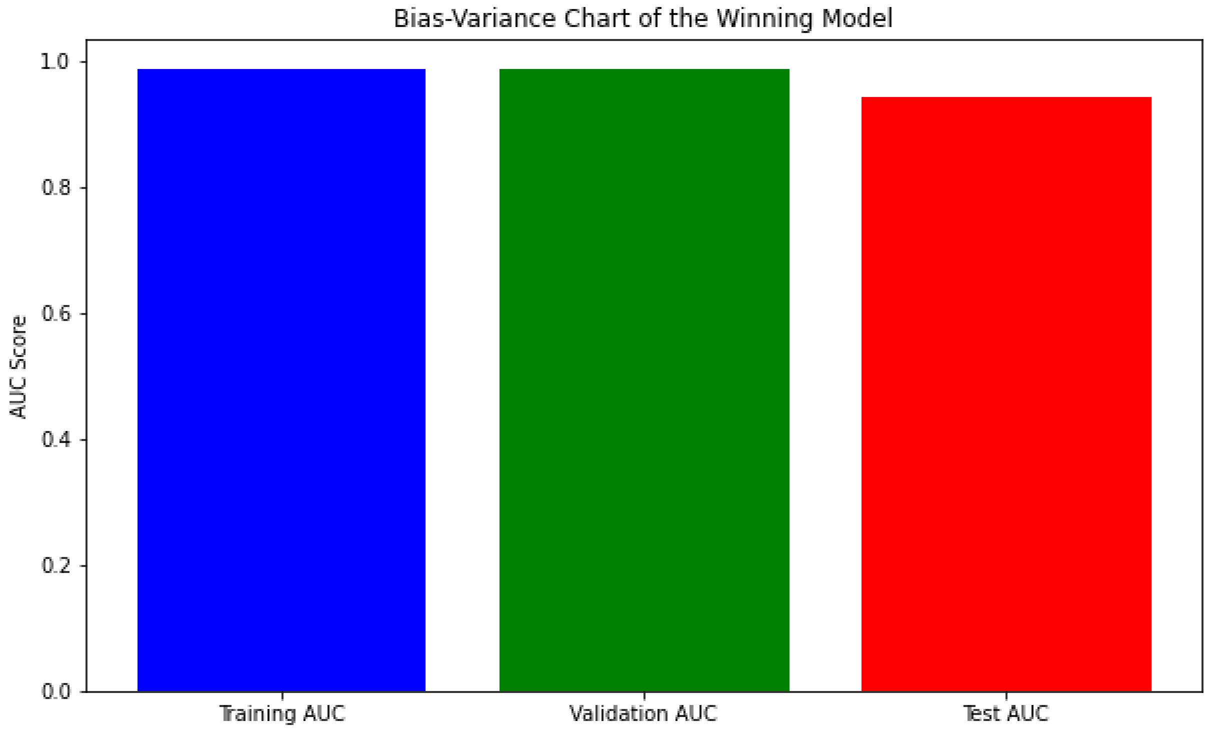
In [63]:
```
# Select the winning model
winning_model_name = max(validation_scores, key=validation_scores.get)
winning_model = {**XGmodel_variations, **LGmodel_variations, **RFmodel_variations}[winn

# Output the performance of the winning model on the test dataset
ytest_pred = winning_model.predict_proba(xtest)[:, 1]
test_auc = roc_auc_score(ytest, ytest_pred)

print(f"Winning model: {winning_model_name} with validation AUC: {validation_scores[win
print(f"Winning model performance on the test dataset: AUC: {test_auc}")
```

```
Winning model: RFmodel_3 with validation AUC: 0.9862408203027578
Winning model performance on the test dataset: AUC: 0.9440211958436756
```

In [64]:
```
# Bias-Variance Trade-off Analysis
# For a full bias-variance analysis, look at the model's performance across multiple tr
# compare training score vs. validation score as a proxy.
train_auc = roc_auc_score(ytrain, winning_model.predict_proba(xtrain)[:, 1])
print(f"Winning model performance on the training dataset: AUC: {train_auc}")
```

```
Winning model performance on the training dataset: AUC: 0.9863891690363809
```

In [65]:
```
# Plotting the Bias-Variance chart
# plotting only AUC scores
plt.figure(figsize=(10, 6))
plt.bar(['Training AUC', 'Validation AUC', 'Test AUC'], [train_auc, validation_scores[w
plt.ylabel('AUC Score')
plt.title('Bias-Variance Chart of the Winning Model')
plt.show()
```



Week 10

In [66]:

```
# Handling missing values for numerical features with median
for column in df2.select_dtypes(include=['float64', 'int64']).columns:
    df2[column].fillna(df2[column].median(), inplace=True)

# Creating new features that might help improve model performance
df2['total_stays'] = df2['stays_in_weekend_nights'] + df2['stays_in_week_nights']

# Split the data into features and target again after the enhancements
x = df2.loc[:, df2.columns != 'is_canceled']
y = df2.loc[:, 'is_canceled']

# Re-split the data into training, validation, and test sets
x_train, x_val_test, y_train, y_val_test = train_test_split(x, y, test_size=0.3, random_
x_val, x_test, y_val, y_test = train_test_split(x_val_test, y_val_test, test_size=0.5,
```

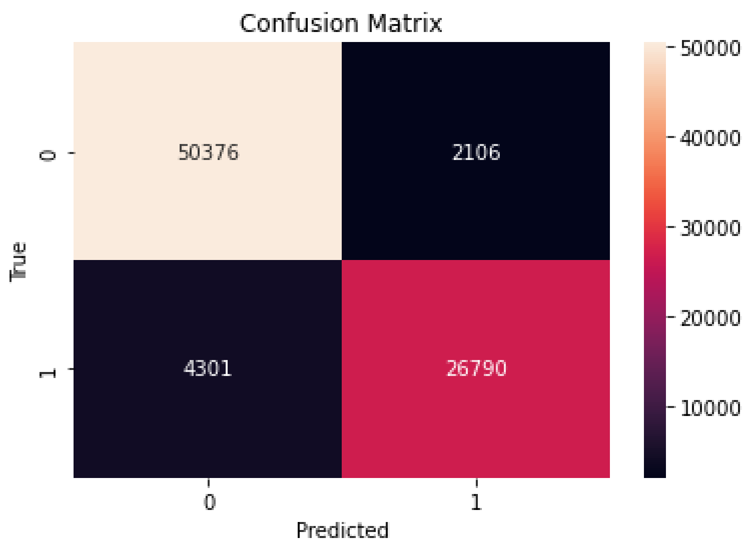
In [67]:

```
# Use RFmodel_3, the winning model as my model
model = RandomForestClassifier(n_estimators=300, max_depth=20, min_samples_split=6, ran
model.fit(x_train, y_train)
y_train_pred = model.predict(x_train)

# Analyze where the model is making errors
errors = y_train != y_train_pred
error_indices = errors[errors].index
error_data = x_train.loc[error_indices]
```

In [68]:

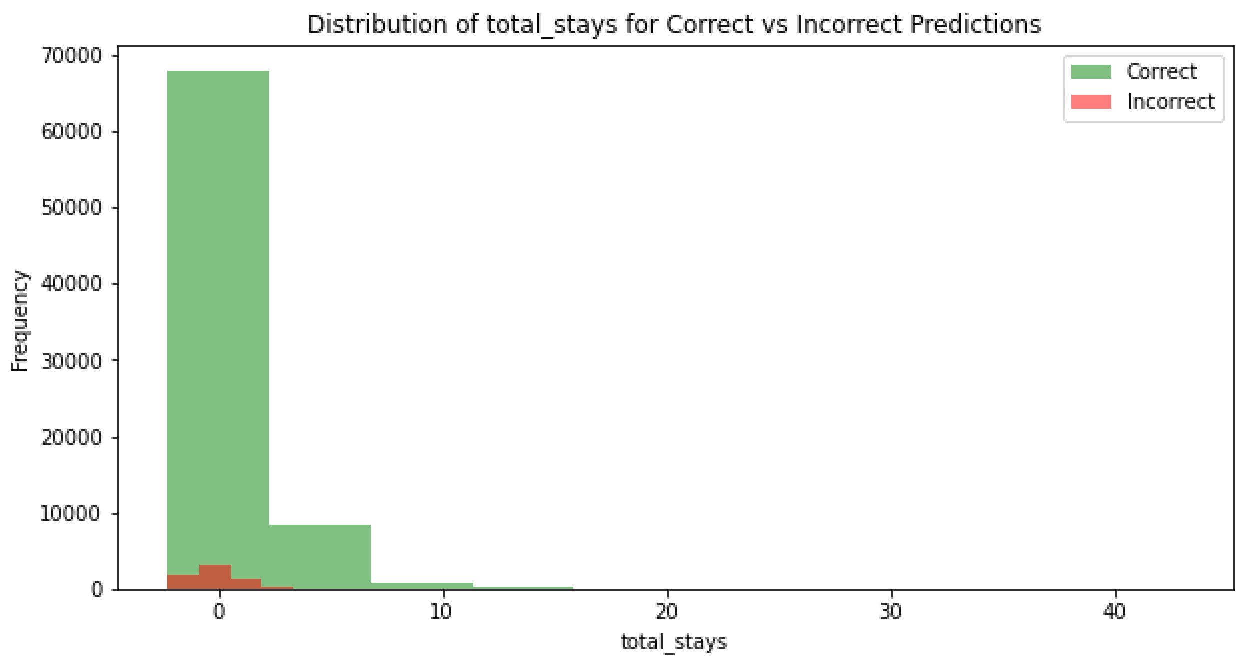
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
cm = confusion_matrix(y_train, y_train_pred)
sns.heatmap(cm, annot=True, fmt='d')
```

```
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```



In [69]:

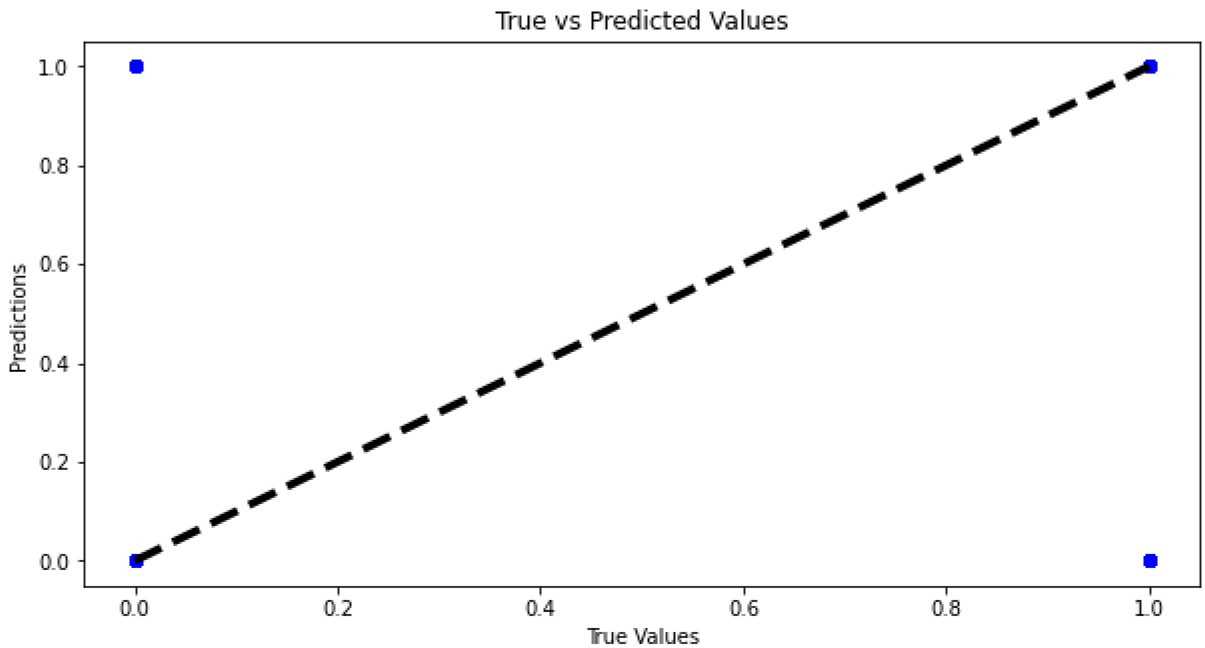
```
feature = 'total_stays'
# Plot the distribution of the feature for correct and incorrect predictions
plt.figure(figsize=(10, 5))
plt.hist(x_train.loc[~errors, feature], color='green', alpha=0.5, label='Correct')
plt.hist(x_train.loc[errors, feature], color='red', alpha=0.5, label='Incorrect')
plt.xlabel(feature)
plt.ylabel('Frequency')
plt.title(f'Distribution of {feature} for Correct vs Incorrect Predictions')
plt.legend()
plt.show()
```



In [71]:

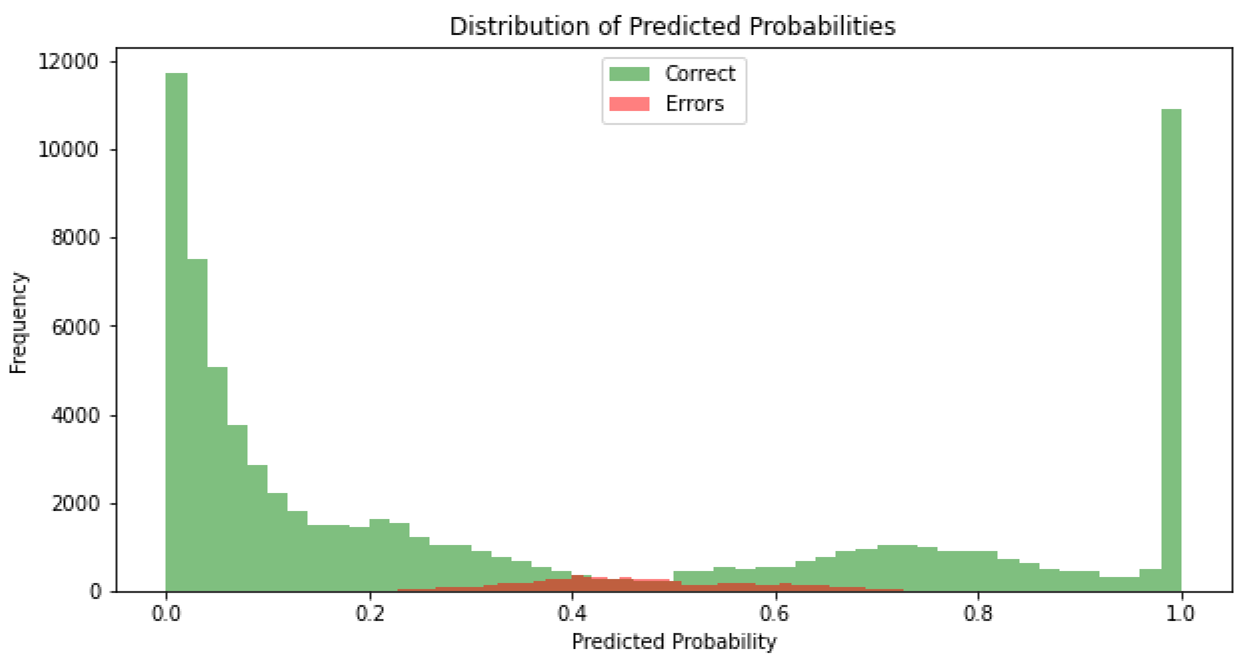
```
plt.figure(figsize=(10, 5))
plt.scatter(y_train, y_train_pred, color='blue')
plt.plot([y_train.min(), y_train.max()], [y_train.min(), y_train.max()], 'k--', lw=4)
```

```
plt.xlabel('True Values')
plt.ylabel('Predictions')
plt.title('True vs Predicted Values')
plt.show()
```



In [72]:

```
# For binary classification
probabilities = model.predict_proba(x_train)[:, 1]
plt.figure(figsize=(10, 5))
plt.hist(probabilities[~errors], bins=50, color='green', alpha=0.5, label='Correct')
plt.hist(probabilities[errors], bins=50, color='red', alpha=0.5, label='Errors')
plt.xlabel('Predicted Probability')
plt.ylabel('Frequency')
plt.title('Distribution of Predicted Probabilities')
plt.legend()
plt.show()
```



In [73]:
```
# Refit the model to the new training dataset
model.fit(x_train, y_train)
```

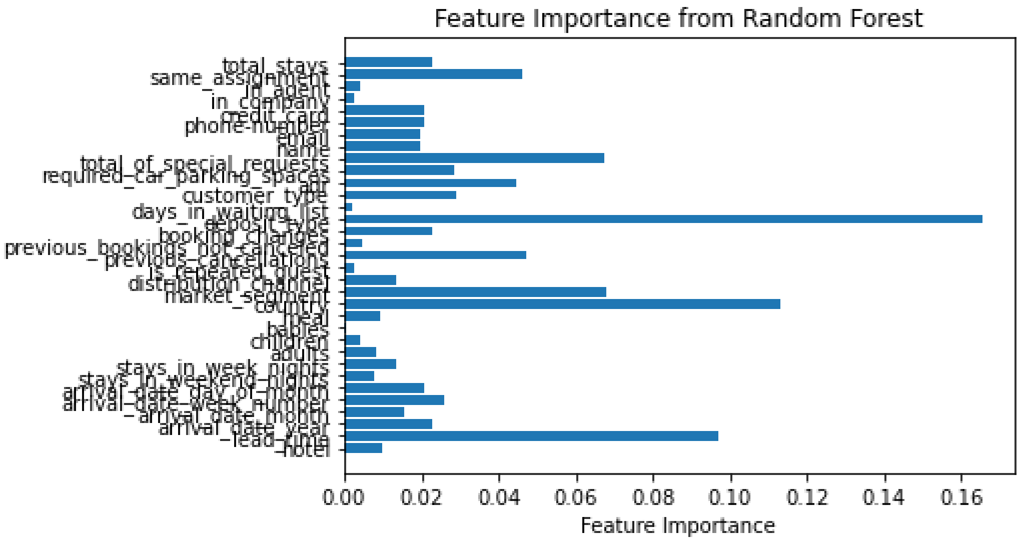
Out[73]:
```
RandomForestClassifier

RandomForestClassifier(max_depth=20, min_samples_split=6, n_estimators=300,
                       random_state=90)
```

In [74]:
```
# Evaluate the model's performance on the new training dataset
y_train_pred = model.predict(x_train)
train_accuracy = accuracy_score(y_train, y_train_pred)

# Evaluate the model's performance on the new validation dataset
y_val_pred = model.predict(x_val)
val_accuracy = accuracy_score(y_val, y_val_pred)

# Evaluate the model's performance on the new test dataset
y_test_pred = model.predict(x_test)
test_accuracy = accuracy_score(y_test, y_test_pred)

# Output the performance
print(f"Training Accuracy: {train_accuracy}")
print(f"Validation Accuracy: {val_accuracy}")
print(f"Test Accuracy: {test_accuracy}")
```

```
Training Accuracy: 0.9233364842712359
Validation Accuracy: 0.8701139155684611
Test Accuracy: 0.8705120330560053
```

Week 11

In [75]:
```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=300, max_depth=20, min_samples_split=6, ran
model.fit(x_train, y_train)

# Get feature importances
importances = model.feature_importances_

# Plot feature importances
plt.barh(range(len(importances)), importances, align='center')
plt.yticks(range(len(importances)), x_train.columns)
plt.xlabel('Feature Importance')
plt.title('Feature Importance from Random Forest')
plt.show()
```



In [77]:

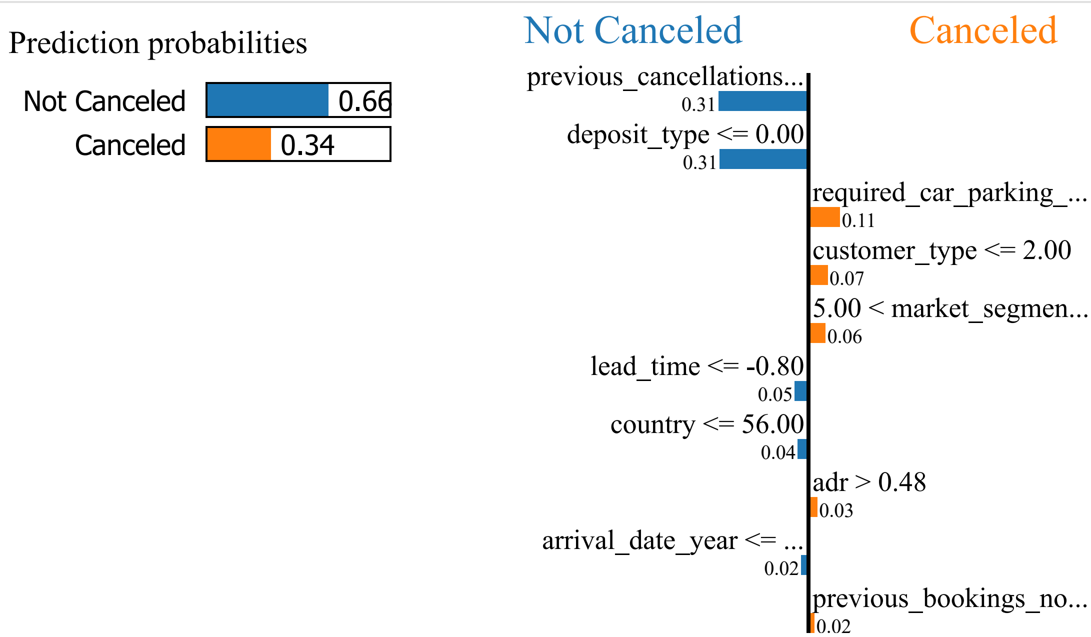
```
!pip install lime
```

```
Collecting lime
  Downloading lime-0.2.0.1.tar.gz (275 kB)
     -------------------------------------- 275.7/275.7 kB 4.3 MB/s eta 0:00:00
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Requirement already satisfied: matplotlib in c:\users\zhumh\anaconda3\lib\site-packages
(from lime) (3.4.3)
Requirement already satisfied: numpy in c:\users\zhumh\anaconda3\lib\site-packages (from
lime) (1.22.4)
Requirement already satisfied: scipy in c:\users\zhumh\anaconda3\lib\site-packages (from
lime) (1.7.1)
Requirement already satisfied: tqdm in c:\users\zhumh\anaconda3\lib\site-packages (from
lime) (4.66.1)
Requirement already satisfied: scikit-learn>=0.18 in c:\users\zhumh\anaconda3\lib\site-p
ackages (from lime) (1.2.2)
Requirement already satisfied: scikit-image>=0.12 in c:\users\zhumh\anaconda3\lib\site-p
ackages (from lime) (0.18.3)
Requirement already satisfied: networkx>=2.0 in c:\users\zhumh\anaconda3\lib\site-packag
es (from scikit-image>=0.12->lime) (2.6.3)
Requirement already satisfied: pillow!=7.1.0,!=7.1.1,>=4.3.0 in c:\users\zhumh\anaconda3
\lib\site-packages (from scikit-image>=0.12->lime) (8.4.0)
Requirement already satisfied: imageio>=2.3.0 in c:\users\zhumh\anaconda3\lib\site-packa
ges (from scikit-image>=0.12->lime) (2.9.0)
Requirement already satisfied: tifffile>=2019.7.26 in c:\users\zhumh\anaconda3\lib\site-
packages (from scikit-image>=0.12->lime) (2021.7.2)
Requirement already satisfied: PyWavelets>=1.1.1 in c:\users\zhumh\anaconda3\lib\site-pa
ckages (from scikit-image>=0.12->lime) (1.1.1)
Requirement already satisfied: cycler>=0.10 in c:\users\zhumh\anaconda3\lib\site-package
s (from matplotlib->lime) (0.10.0)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\zhumh\anaconda3\lib\site
-packages (from matplotlib->lime) (2.8.2)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\zhumh\anaconda3\lib\site-pa
ckages (from matplotlib->lime) (1.3.1)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\zhumh\anaconda3\lib\site-pac
kages (from matplotlib->lime) (3.0.4)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\zhumh\anaconda3\lib\site
-packages (from scikit-learn>=0.18->lime) (2.2.0)
Requirement already satisfied: joblib>=1.1.1 in c:\users\zhumh\anaconda3\lib\site-packag
es (from scikit-learn>=0.18->lime) (1.2.0)
```

```
Requirement already satisfied: colorama in c:\users\zhumh\anaconda3\lib\site-packages (f
rom tqdm->lime) (0.4.4)
Requirement already satisfied: six in c:\users\zhumh\anaconda3\lib\site-packages (from c
ycler>=0.10->matplotlib->lime) (1.16.0)
Building wheels for collected packages: lime
  Building wheel for lime (setup.py): started
  Building wheel for lime (setup.py): finished with status 'done'
  Created wheel for lime: filename=lime-0.2.0.1-py3-none-any.whl size=283857 sha256=accc
7f1c60d7a57c10cb11b5a95a4f0e8cbd2041320e8679ce3e7a8767a2a537
  Stored in directory: c:\users\zhumh\appdata\local\pip\cache\wheels\ed\d7\c9\5a0130d06d
6310bc6cbe55220e6e72dcb8c4eff9a478717066
Successfully built lime
Installing collected packages: lime
Successfully installed lime-0.2.0.1

[notice] A new release of pip is available: 23.0 -> 23.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

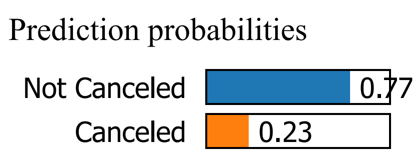
In [78]:

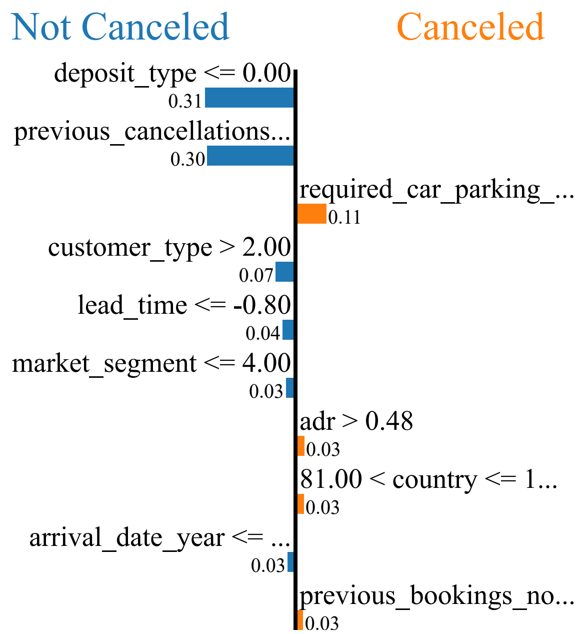
```python
from lime import lime_tabular

explainer = lime_tabular.LimeTabularExplainer(
    training_data=np.array(x_train),
    feature_names=x_train.columns,
    class_names=['Not Canceled', 'Canceled'],
    mode='classification'
)

# Select 5 random instances from the test set
np.random.seed(90)
random_indices = np.random.choice(x_test.index, size=5, replace=False)

# Explain each of the 5 instances
for i in random_indices:
    exp = explainer.explain_instance(x_test.loc[i], model.predict_proba, num_features=1
    exp.show_in_notebook(show_table=True, show_all=False)
```



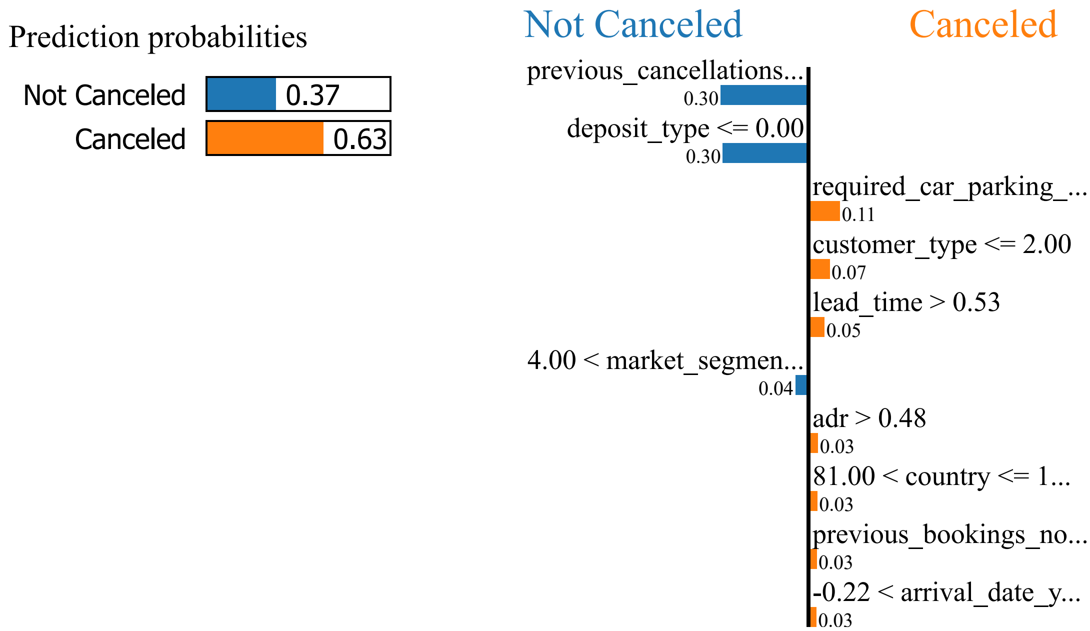
Feature	Value
previous_cancellations	-0.10

deposit_type	0.00
required_car_parking_spaces	-0.25
customer_type	2.00
market_segment	6.00
lead_time	-0.92
country	1.00
adr	1.05

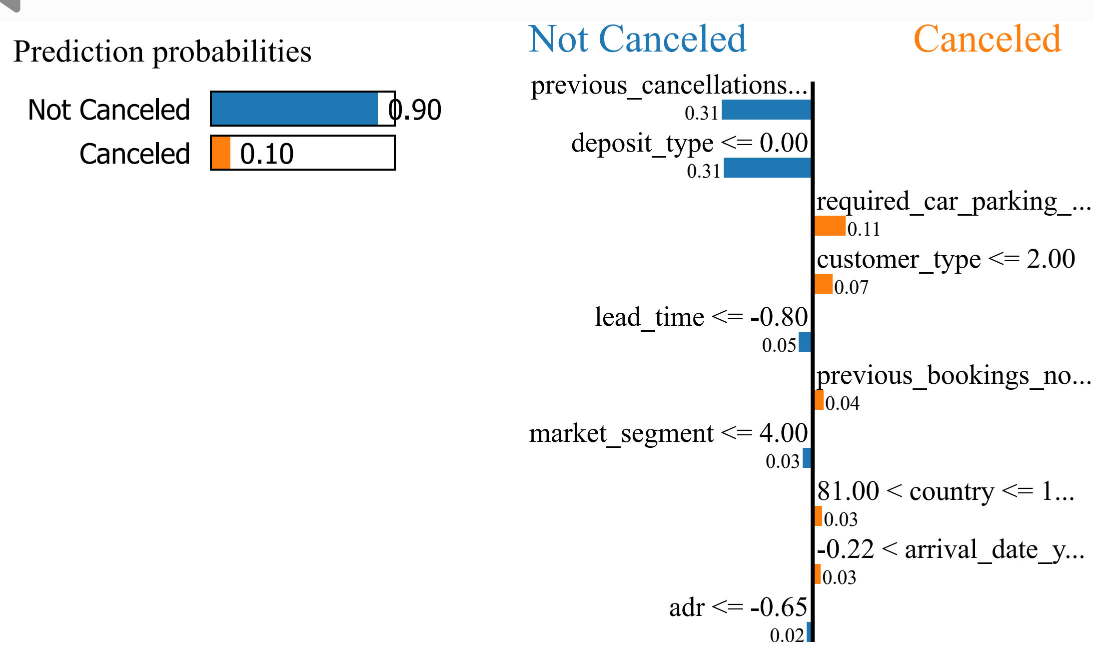




Feature	Value
deposit_type	0.00
previous_cancellations	-0.10
required_car_parking_spaces	-0.25
customer_type	3.00
lead_time	-0.95
market_segment	2.00
adr	0.74
country	135.00
arrival_date_year	-0.22

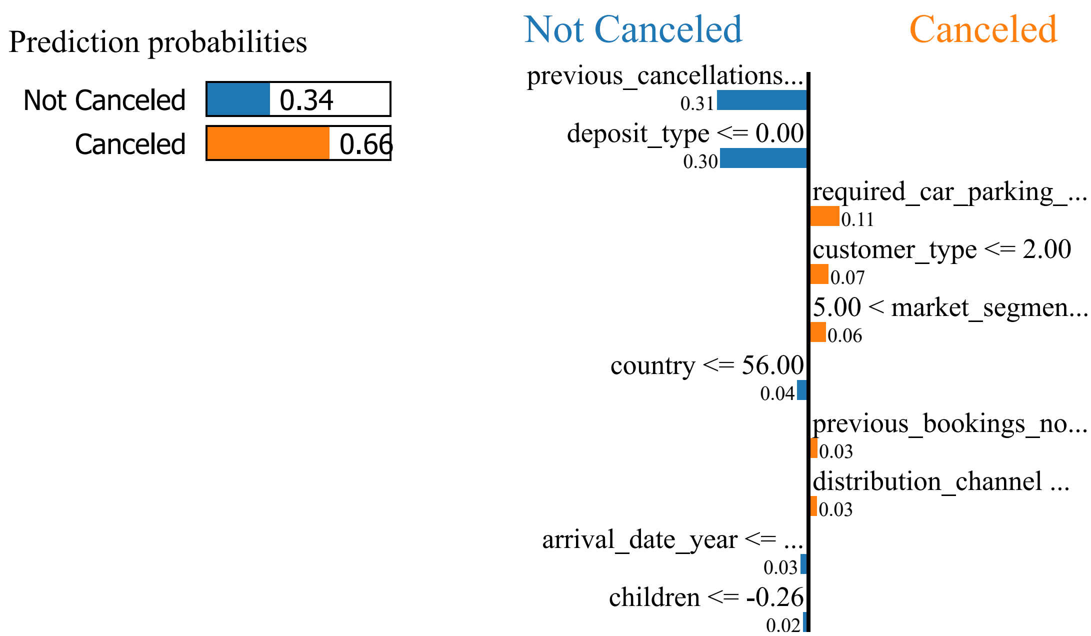


Feature	Value
previous_cancellations	-0.10
deposit_type	0.00
required_car_parking_spaces	-0.25
customer_type	2.00
lead_time	0.67
market_segment	5.00
adr	0.99
country	135.00
previous_bookings_not_canceled	-0.09



Feature	Value

previous_cancellations	-0.10
deposit_type	0.00
required_car_parking_spaces	-0.25
customer_type	2.00
lead_time	-0.88
previous_bookings_not_canceled	-0.09
market_segment	2.00
country	135.00



Feature	Value
previous_cancellations	-0.10
deposit_type	0.00
required_car_parking_spaces	-0.25
customer_type	2.00
market_segment	6.00
country	43.00
previous_bookings_not_canceled	-0.09
distribution_channel	3.00
arrival_date_year	-0.22

In [82]:

```
# define the subgroups based on the country column. It's a list of countries from the b
countries = ['PRT', 'GBR', 'ESP', 'IRL', 'FRA', 'DEU', 'BEL', 'ITA', 'NLD', 'BRA']

# Create a dictionary to hold the data for each subgroup
subgroups = {country: df[df['country'] == country] for country in countries}
```

```
# perform analyses on each subgroup, such as calculating the cancellation rate
cancellation_rates = {}

for country, subgroup in subgroups.items():
    cancellation_rate = subgroup['is_canceled'].mean()
    cancellation_rates[country] = cancellation_rate

# Output the cancellation rates for each country
print("Cancellation rates by country:")
for country, rate in cancellation_rates.items():
    print(f"{country}: {rate:.2f}")
```

```
Cancellation rates by country:
PRT: 0.57
GBR: 0.20
ESP: 0.25
IRL: 0.25
FRA: 0.19
DEU: 0.17
BEL: 0.20
ITA: 0.35
NLD: 0.18
BRA: 0.37
```

In [87]:

```
median_stay = df2['total_stays'].median()

subgroups = {
    'Short Stays': df2[df2['total_stays'] <= median_stay],
    'Long Stays': df2[df2['total_stays'] > median_stay]
}
```

In [88]:

```
from sklearn.metrics import accuracy_score

# Calculate accuracy for each subgroup
bias_analysis = {}
for subgroup_name, subgroup_data in subgroups.items():
    x_subgroup = subgroup_data.drop(['is_canceled'], axis=1)
    y_subgroup = subgroup_data['is_canceled']
    y_pred_subgroup = model.predict(x_subgroup)
    accuracy_subgroup = accuracy_score(y_subgroup, y_pred_subgroup)
    bias_analysis[subgroup_name] = accuracy_subgroup

# Print the bias analysis results
for subgroup_name, accuracy in bias_analysis.items():
    print(f"{subgroup_name} Accuracy: {accuracy}")
```

```
Short Stays Accuracy: 0.9178891404942404
Long Stays Accuracy: 0.8965234054202026
```

In [89]:

```
# Strategies might include resampling, reweighting, or feature engineering, reweight the
from sklearn.utils import class_weight

class_weights = class_weight.compute_class_weight(
    'balanced',
    classes=np.unique(y_train),
    y=y_train
)
```

```
model.set_params(class_weight=dict(enumerate(class_weights)))
model.fit(x_train, y_train)
```

Out[89]:

```
RandomForestClassifier

RandomForestClassifier(class_weight={0: 0.7962063183567699,
                                     1: 1.3440063040751342},
                       max_depth=20, min_samples_split=6, n_estimators=300,
                       random_state=90)
```

In [90]:

```
# Evaluate the debiased model
y_train_pred_debiased = model.predict(x_train)
y_val_pred_debiased = model.predict(x_val)

# Compare accuracy
print('Training Accuracy:', accuracy_score(y_train, y_train_pred_debiased))
print('Validation Accuracy:', accuracy_score(y_val, y_val_pred_debiased))
```

```
Training Accuracy: 0.9348473789381738
Validation Accuracy: 0.8730176457449185
```